NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

BOILERMODEL: A QUALITATIVE MODEL-BASED REASONING SYSTEM IMPLEMENTED IN ADA

by

James F. Stascavage

September 1991

Thesis Advisor: Prof. Yuh-jeng Lee

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
BOILERMODEL: A QUALITATIVE MODEL-BASED REASONING SYSTEM IMPLEMENTED IN ADA (U)

12. PERSONAL AUTHOR(S)
Stascavage, James F.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 09/89 TO 09/91	September 1991	142

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Model-based Reasoning, Qualitative Modeling, Naval Engineering, Naval Training, Ada in Artificial Intelligence

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Effective, inexpensive, and realistic on-going training is required to keep all Naval personnel proficient in their fields. Nowhere is this more true than in steam propulsion engineering plants. The complex systems of valves, piping, and components require continual refresher for watchstanders to perform their jobs safely.

BoilerModel is a qualitative expert system designed using model-based reasoning principles and implemented in Ada. It accurately models a 1200 psi D-type boiler and its associated peripherals. The use of fundamental intra-component relationships ("first principles") and constraint propagation result in compact code because there is no need for the extensive rule base found in conventional expert systems. Implementation in Ada permits the use of concurrent tasking to simulate simultaneous valve propagation found in real-world boiler systems. Additionally Ada's portability allows BoilerModel to be compiled and run on virtually any machine, thereby making it an affordable and attractive complement to shipboard engineering training.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Yuh-jeng Lee	22b. TELEPHONE (Include Area Code): (408) 646-2361	22c. OFFICE SYMBOL: CS/Le

Approved for public release; distribution is unlimited

BOILERMODEL: A QUALITATIVE MODEL-BASED REASONING SYSTEM IMPLEMENTED IN ADA

by

James F. Stascavage
Lieutenant, U.S. Navy
B.A., University of Dallas, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
26 September 1991

ABSTRACT

Effective, inexpensive, and realistic on-going training is required to keep all Naval personnel proficient in their fields. Nowhere is this more true than in steam propulsion engineering plants. The complex systems of valves, piping, and components require continual refresher for watchstanders to perform their jobs safely.

BoilerModel is a qualitative expert system designed using model-based reasoning principles and implemented in Ada. It accurately models a 1200 psi D-type boiler and its associated peripherals. The use of fundamental intra-component relationships ("first principles") and constraint propagation result in compact code because there is no need for the extensive rule base found in conventional expert systems. Implementation in Ada permits the use of concurrent tasking to simulate simultaneous valve propagation found in real-world boiler systems. Additionally, Ada's portability allows BoilerModel to be compiled and run on virtually any machine, thereby making it an affordable and attractive complement to shipboard engineering training.

TABLE OF CONTENTS

I. INTRODUCTION

The technical nature of most U.S. Navy jobs requires a substantial investment (in terms of man-hours lost, equipment maintenance, materials, etc.) for initial training. On-going training is also required to sustain a satisfactory level of proficiency. There is, therefore, always a need for effective, realistic, and inexpensive complements to conventional schooling to maintain competency. Nowhere is this more true than for the training of steam propulsion engineering plant operators. The complex, almost Gordian knot of valves, piping, and components is overwhelming to the novice and requires *continual* refresher for qualified watchstanders to perform their jobs effectively and safely. However, the Navy currently has only one computerized steam plant simulator (the Propulsion Plant Trainer (PPT) in Newport, R.I.) and one non-specific stationary hot plant (at Great Lakes Naval Station). "Hands-on" training for prospective division officers and department heads is conducted at one of these two facilities, or onboard ships moored to a pier.

A. STEAM ENGINEERING TRAINING PROBLEMS

Three problems are evident with the status quo. First, hands-on training focuses on *proper* (i.e., non-catastrophic) operation of the plant. With the exception of the PPT, it is too dangerous to both machinery and human life to impose actual casualty situations on steaming boilers. Therefore, most casualty control training is either learned in the classroom or is simulated. (Simulated casualty control is like kissing one's own sister; it isn't quite the same thing).

The second problem is that training platforms are expensive to maintain. Machinery at the hot plant and onboard ships breaks. The PPT undergoes physical changes to match

real-world ship alterations, and these changes often require software updates. Additionally, building a PPT for the West Coast (to fill the training gap) would be a multimillion dollar expenditure. Both the hot plant and "school ships" burn fuel while training. This fuel could be better used getting the ships and their crews underway conducting at-sea operations (where they should be in the first place).

The third problem is that plant line-up changes and casualty restoration is very time consuming. With the exception of the PPT (where restoration is instantaneous), prospective engineering officers spend much of their time on the deckplates answering questions from the instructors and *not* learning by doing. While this problem is non-existent in the PPT, there is only one PPT. The few steam ships stationed in Newport are virtually the only ones that can afford to send watch teams to the trainer.

B. RESEARCH QUESTIONS

The problems with current on-going fleet steam engineering training form the background for the following questions posed by this thesis.

First, can an expert system be developed that effectively and efficiently models boiler operation? If so, can it be designed in such a manner that it can be expanded to model the entire steam plant?

Second, can such a model be constructed using qualitative reasoning such that it is not limited by parameters and features specific to one platform?

Third, must a model-based expert system be written in Lisp or one of the other traditional artificial intelligence languages, or can it be written in a general purpose language such as Ada?

Fourth, can such a system be made inexpensively enough to make it an attractive and affordable shipboard tool?

BoilerModel was developed as part of this thesis to answer the questions posed. It is a fairly uncomplicated qualitative model-based reasoning system whose domain is the naval propulsion boiler. It is implemented in Ada. The cause-effect propagation of events in the model-based paradigm is ideally suited for physical applications such as steam generation plants. Model-based systems are beneficial in education and training because they can progress through events causally in much the same manner as students learn. They rely on how components work and how they are interrelated. Thus, plant scenarios can be generated easily by students and abnormal conditions can be diagnosed confidently by watchstanders.

C. THESIS OUTLINE

The second chapter of this thesis provides the reader with an introduction to model-based reasoning and the differences between it and rule-based inference. The third chapter serves as a literature review or survey of related work in the fields of qualitative physics and model-based reasoning. The fourth chapter introduces the model domain (the propulsion boiler system), including important cause-effect relationships. The fifth chapter focuses on BoilerModel as an Ada implementation. The questions posed in this introductory chapter will be answered in the body of the thesis; synopses of the answers will be provided in the concluding remarks of the sixth chapter.

II. A MODEL-BASED REASONING PRIMER

Expert systems today can be categorized in two general ways: as rule-based systems or as model-based systems. Of course, designs may be hybrids, containing elements of both. Rule-based systems consist of sets of known facts and rules in the problem domain. These elements come from interviews with subject matter experts and domain-specific technical documentation. An *inference engine,* some sort of program or production language uses the rules and facts in the knowledge base to reason from input. Rule-based systems are wholly dependent on the facts and relationships in the knowledge base; therefore, the more facts and rules, (generally) the more robust the expert system. Model-based systems approach the problem from a different tack. This chapter will examine what model-based reasoning is, how it works and what utility it has in problem solving situations.

A. REASONING FROM MODELS

Model-based expert systems have been written in many languages and for many different architectures. Knowledge representation also differs from system to system to suit the specifications of the designers and the needs of the users. However, all of these systems have one thing in common: they reason from some sort of model of the domain. While a rule-based system may reason exclusively from observed values to facts or rules in its knowledge base, model-based systems reason from "first principles," rules which describe the internal processes and causal relationships between components in the domain. Since first principles are facts about objects and how they behave, they can reason from observed values to real-world states simply by generating different system states,

propagating these constraints through the first principles, and comparing the generated sensor values with actual observed values.

> "The essence of [the] model-based expert system approach is to generate a model that acts as close to the real world as possible except when a measurement or component fails. . . .When the real world begins to act differently from the model, we detect the discrepancy and diagnose the change using the model." (Fulton and Pepe, 1990, pp. 52-3)

The Rube Goldberg drawing in Figure 2.1 (Kinnaird, 1968, p. 37) lightheartedly shows the internal states and cause-effect relationships required to build a better mousetrap.

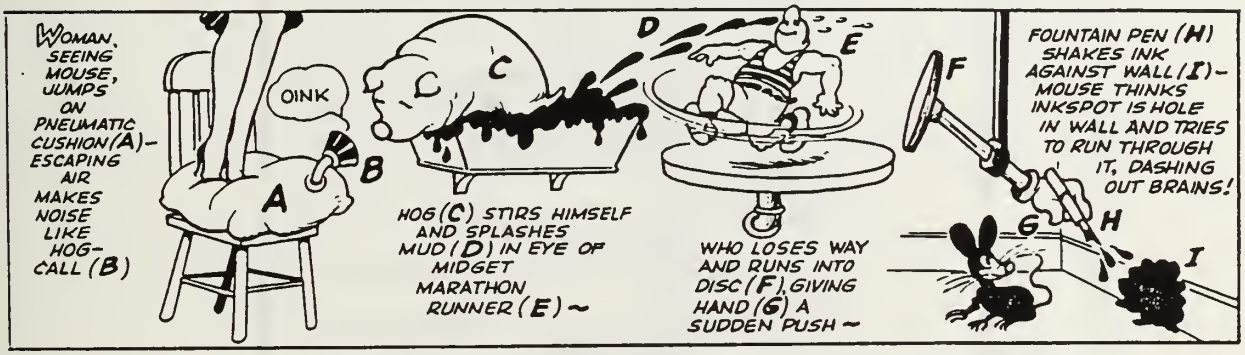


Figure 2.1

The heart of the model is constraint propagation, which is to the model-based reasoning system what the inference engine is to the rule-based system. Propagation uses the relationships between components to establish a chain reaction when changes are made to the system. Propagation continues to occur until all valid relationships have been

explored. For example, consider the simple valve and piping arrangement in Figure 2.2 and the corresponding valid set of steam pressure propagation relationships in Table 2.1

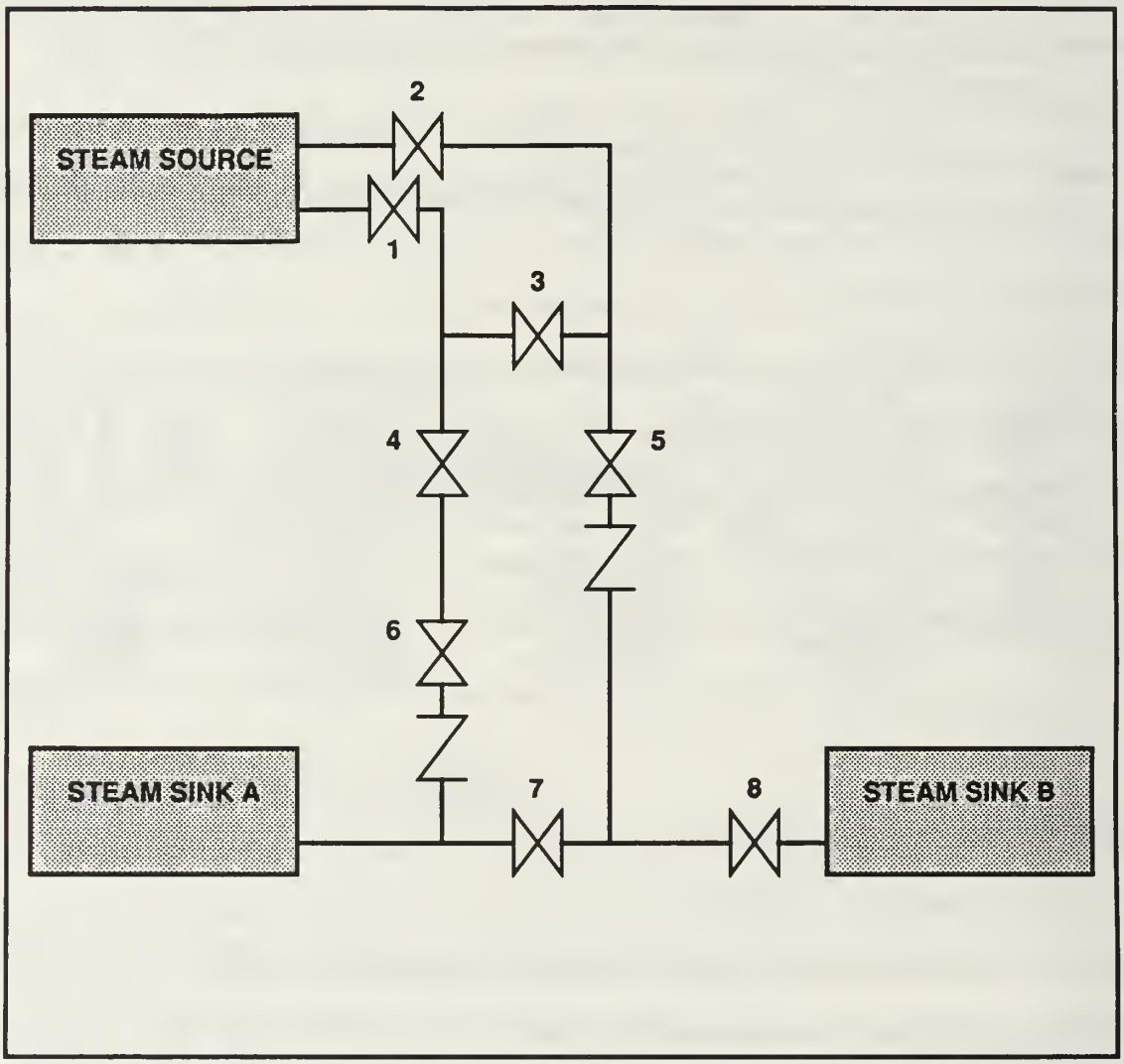


Figure 2.2

```
VALVE 1 input = NORM
VALVE 1 output = VALVE 1 input (if open) /
                 NONE (if shut)
VALVE 2 input = NORM
VALVE 2 output = VALVE 2 input (if open) /
                 NONE (if shut)
VALVE 3 input = greater of (VALVE 1 output,
                            VALVE 2 output)
VALVE 3 output = VALVE 3 input (if open) /
                 NONE (if shut)
VALVE 4 input = greater of (VALVE 2 output,
                            VALVE 3 output)
VALVE 4 output = VALVE 4 input (if open) /
                 NONE (if shut)
VALVE 5 input = greater of (VALVE 1 output,
                            VALVE3 output)
VALVE 5 output = VALVE 5 input (if open) /
                 NONE (if shut)
VALVE 6 input = VALVE 4 output
VALVE 6 output = VALVE 6 input (if open) /
                 NONE (if shut)
VALVE 7 input = greater of (VALVE 5 output,
                            VALVE 6 output)
VALVE 7 output = VALVE 7 input (if open) /
                 NONE (if shut)
VALVE 8 input = greater of (VALVE 5 output,
                            VALVE 7 output)
VALVE 8 output = VALVE 8 input (if open) /
                 NONE (if shut)
STEAM SINK A value = greater of (VALVE 6 output,
                                 VALVE 7 output)
STEAM SINK B value = VALVE 8 output
```

Table 2.1

Reasoning about what effect shutting VALVE 1, VALVE 3, and VALVE 7 has on the values of STEAM SINK A and STEAM SINK B would simply be a matter of changing the status of those valves and reevaluating the relationships.

Note also that the rules, although specific to the valves, effectively simulate the function of the piping system. Moreover, the entire piping system is modeled. How well a model simulates real-world connectivity, of course, depends on its applications. In

general, though, the more representative rules are of the relationships between components, the more robust the overall model will be and the more diversified its potential applications.

Models themselves fall into two broad groups: quantitative and qualitative. Quantitative models fall outside the scope of this research. Briefly, quantitative models rely on mathematical or numerical rules and relationships to predict or monitor system functions.

> "A mathematical model could be constructed to model the function of a grandfather clock, taking into account the oscillator length, gear size, and so on. A numeric model could predict the position of the hands after a specific time interval." (Fulton and Pepe, 1990, p. 51)

Qualitative models, on the other hand, describe domain components "in terms of causal, compositional or subtypical relationships among objects and events." (Clancey, 1989, p. 10) There are several variations of qualitative models. *Classification models* categorize observed patterns to describe processes. The process descriptions identify events which occur over time and in diverse locations. Diagnosing infectious diseases is one example of the use of classification models. *Simulation models* start from a set of initial conditions and predict how the systems will change when the initial conditions are changed. *Functional models* relate system behaviors and states to functional goals. (Clancey, 1989, p. 13) White and Frederiksen (1989) discuss *phenomenological* and *reductionist models* (see Chapter III.C.1).

B. MODEL-BASED vs. RULE-BASED SYSTEMS

As discussed previously, rule-based systems are wholly dependent on facts and rules in their knowledge bases. They cannot, in and of themselves, reason from cause to effect unless the cause and effect happen to be rules accessible to the inference engine. Model-based systems can because cause-effect relationships are easily and naturally modeled as first principles. This is especially important in applications involving physical systems such as steam generation plants and electrical distribution systems. "Rule-based expert

systems were never particularly suited to industrial monitoring applications." (Fulton and Pepe, 1990, p. 48) Reasons for this fall into three general areas.

1. Sensor Failure

Control personnel in real-world industrial systems rely on information from sensors to formulate decisions or perform diagnostics. A rule-based system would require a set of rules mapping possible sensor readings to corresponding plant conditions. A problem arises in that sensor indicators (such as thermometers, pressure gauges, etc.) can themselves fail on occasion. A rule-based system would then be required to have in its knowledge base a set of rules which would ascertain for any sensor reading whether or not that data is correct. "According to current estimates, up to three-quarters of industrial expert system rules do nothing more than verify sensor accuracy." (Fulton and Pepe, 1990, p. 49).

Model-based systems, on the other hand, have only as many component description and systems interrelationships as are necessary to define the domain. Out-of-limits sensor readings due to faulty sensors can be accurately diagnosed in exactly the same manner as out-of-limits readings due to plant malfunction: components upstream of the sensor in the model are failed in various combinations until a match between model sensor values and real-world sensor values is obtained. If the only match(es) between model and actual system contain contradictory component state information, then the sensor must be faulty (because it is assumed that the real-world system has been accurately and completely modeled). In the simple valve and piping example introduced previously, if the sensor for STEAM SINK A indicates a value of NONE, the expert system would only be able to propagate to that result if certain combinations of valves were shut. The set of such combinations is finite, and if no element of that set matches the known valve line-up, then the sensor is determined to be faulty and in need of recalibration or replacement.

2. Number of Rules

The sheer number of rules needed to correctly predict plant performance or diagnose faults in systems of even moderate size is enormous. Consider again the valve and piping example. Given eight valves, each having a status of either open or shut, and a constant value coming from the STEAM SOURCE, a rule-based system could require up to 256 rules of the form in Figure 2.3.

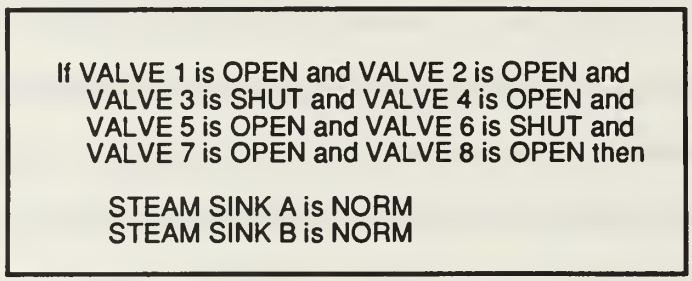


Figure 2.3

This plethora of rules presents four problems which are resolved when model-based systems are used. First, as the number of rules/facts increases, the chances of implementing an exhaustive rule base decreases. "A traditional expert system relies on expert experience likely to be deepest concerning common failures; the uncommon failure is doomed to obscurity and may not be properly diagnosed." (Fulton and Pepe, 1990, p. 55) During the interview process, the expert may not remember or even be familiar with obscure casualty situations which may occur. The completeness and accuracy of the rule-based system is very dependent on the experience of the experts and the questions posed by the designers. Since the model-based approach is founded on first principles which describe component behavior and are essentially independent of expert experience, this problem is obviated. Large components can be broken down into smaller actual or virtual

components to the point where cause-effect relationships are manageable, accurate, and as complete as necessary.

Second, in a large rule-base there may exist some rules which contradict each other, or in concert with each other produce inaccurate results. There may also be rules which are just not correct. "As the number of objects in the system increases, it becomes very difficult for the expert to predict accurately what state each sensor will be in for each possible failure." (Fulton and Pepe, 1990, p. 49) Moreover, determining which rules failed in a particular situation can be very difficult because there is no necessary link between inter- /intra-component behavior and knowledge base elements. A model-based system's network of behaviors, because it focuses first on component or subcomponent behavior and then on relationships, does not grow increasingly more complex as the modeled system grows (although the number of components and inter-component relationships that must be modeled does grow). Additionally, pinpointing faulty device behaviors or relationships is as simple as tracing through the state tables for a device as values propagate. Determining where errors existed in this project was never a problem; fixing them, of course, was a different story.

Third, a large rule base is expensive in terms of time spent in development. Since such a system would require extensive contact between design personnel and subject matter experts, there would exist a large period of time in which the expert system was in production. Additionally, as the real-world system changes, experts (who do not work for free) would have to be consulted for modifications to the rule base. Although some time lag between conception and implementation would also exist for a model-based system, picking the brains of experts for facts or rules to support all contingencies is unnecessary. Only when new components (which have not been previously modeled) are added will there be a substantial time drain. "Since rules are not created . . . the component knowledge

base simply is modified to reflect the changes, [and] a task is no more difficult than altering a set of schematics." (Fulton and Pepe, 1990, p. 55)

Fourth, the addition of new components in rule-based systems increases exponentially the number of new rules needed. In the now familiar valve and piping example, adding a ninth valve would increase the exhaustive rule set from 256 to 512 (or 2^9) if a rule-based system were used. Changes to a model-based system would be limited to information about that valve's input and output and changes to the valves immediately upstream and downstream of it (effectively re-linking the system).

3. Human Expert

Model-based systems more closely simulate how human experts diagnose faults or predict system behavior. When there is incomplete or conflicting information available, human experts rely on what data is available and formulate hypotheses upon which future actions (repair work, casualty control measures, etc.) are based.

> "Generating rules to compensate for even a subset of the possible partial data situations is an onerous task that catastrophically increases the required number of rules. It is highly improbable such an effort will ever provide complete scenario coverage." (Fulton and Pepe, 1990, p. 49)

Model-based reasoning closely approximates the cause-effect reasoning mechanism employed in human learning. The study of mathematics and science is fraught with facts and figures which are used in problem solving (a cause-effect exercise). The non-quantitative world is also understood analytically. A foreigner unfamiliar with baseball will learn the game more quickly by watching (and doing) than by just memorizing facts and rules.

C. UTILITY OF MODEL-BASED SYSTEMS

Model-based systems are ideally suited for industrial, system oriented applications such as electrical circuitry training and steam plant monitoring. In applications such as

these, model-based systems can be conceptually true to real-world configurations without causing immense complications during real-world modifications. Model-based systems are well suited for use in the realm of training and education. "When novices spontaneously attempt to understand how physical systems work, they use constructs such as 'causality,' 'mechanism,' and 'purpose'." (White and Frederiksen, 1989, p. 84) Models are also well suited for fault diagnosis; artificially "failing" components and propagating the new values in order to match the abnormal real-world status can be done quickly and will exhaust all casualty conditions in a well-modeled system.

Model-based expert systems are not well suited for domains in which a complete and accurate model cannot be created. Poor quality cause-effect relationships result in poor quality diagnoses and predictions which may endanger life and equipment. Model-based systems also require real-world sensors to provide enough information for reasonably accurate diagnoses. If such sensors are missing and cannot be installed, only partially accurate and downright useless conclusions will be formulated by the expert system, and a rule-based system may be the best bet.

III. LITERATURE REVIEW

A. INTRODUCTION

Qualitative model-based reasoning systems have their foundations in the qualitative physics/commonsense reasoning pioneered in the late 1970's and early 1980's by de Kleer, Brown, and Forbus (Iwasaki, 1989). Reasoning systems based on cause-effect relationships were developed during the same period, and some systems have evolved beyond the drawing board and are in use today. The end use of model-based reasoning systems to date have been in the areas of intelligent tutoring systems and fault diagnosis. Several systems ranging from the pragmatically spartan to the graphically complex will be examined here.

B. QUALITATIVE PHYSICS FOUNDATIONS

Three distinct yet complementary ways of qualitatively explaining the effects of physical laws on systems and devices were developed in the late 1970's and early-to-mid 1980's. They focused on devices themselves (ENVISION), processes between devices (Qualitative Process Theory), and system behavior (QSIM). Additionally, World Qualitative Modeling System (WQMS), published in 1990, uses elements of all three camps. The four systems will be discussed in this section.

1. ENVISION

ENVISION was developed by de Kleer and Brown at the Xerox Palo Alto Research Center (Iwasaki, 1989). It takes a device or component centered view of a system; the system as an entity consists as an integration of many thoroughly specified and described component parts. Of primary concern are the individual devices and their interconnections. To this end, the devices have to be isolated and their functions carefully

defined to infer nothing about the workings of the system in which they are located. These two considerations are more formally known as the Locality Principle and the No-function-in-structure Principle. Although it is impossible to adhere strictly to these principles when describing a component, they serve as ideals toward which the description should be aimed (Iwasaki, 1989, p. 363).

The connections between devices, called conduits, are assumed to transmit information instantaneously and equally when there are multiple conduits between components.

Device behavior is divided into inter-state and intra-state behaviors and is predicted using the qualitative functions (equations) in the definition of the device. Prediction is based on propagating known values through the equations of the device, producing both a logical cause/effect link and new facts. Once the intra-state behavior of the device has been determined, all possible future states of that device can be taken from a table that is indexed by the values of state variables and contains all legal states for that device. Prediction of system behavior, then, is an exercise in determining all logical transitions between the possible future states along conduits.

2. QPT

Qualitative Process Theory (QPT) was developed in 1982 by Forbus at Massachusetts Institute of Technology (Iwasaki, 1989). His Qualitative Process Engine uses information about objects and processes to reason about which processes will occur, what they will affect in the system, and when they will stop (Iwasaki, 1989, p. 371). Physical systems are represented as objects which have certain defined interrelationships and processes which are the sole means of changing state in the system. Examples of processes in QPT are heat flow, boiling, and evaporation.

Individual views in QPT are "views of objects that focus on the enabling conditions of their behavioral characteristics" (Iwasaki, 1989, p. 372). For example, *water* in a boiler and water flowing through a nozzle would have different individual views because the same object, water, behaves differently in the two situations. Individual views have four parts:

(1) the individual object(s) involved;

(2) the quantity condition, which is a statement defining quantities of individuals, generally with respect to each other;

(3) preconditions, which are conditions other than quantity conditions which must hold true for the individual view to be valid. For example, in order for a container to hold a liquid, the container must be capable of holding a liquid (Iwasaki, 1989, p. 373);

(4) relations, which are further truths regarding attributes of the individual(s). For example, the individual view of water in a nozzle would have a relationship (mass flow rate, or $\dot{m}$) between the specific density of the water (ρ), its velocity ($\bar{v}$), and the cross-sectional area of the nozzle (A):

$$\dot{m} = \rho \times A \times \bar{v} \qquad \text{(eq 3.1)}$$

Processes must include the individuals necessary for the process to be activated, what events/circumstances will trigger the process, and what changes will be brought about when a process has been run.

Behavior of a system based on a given set of objects and their relationships can be predicted. Individual views are created and relevant processes are activated. Processes can then start and stop, creating new individual views and removing antiquated ones. The

Qualitative Process Engine "detects the processes that must take place in a given situation and predicts their course." (Iwasaki, 1989, p. 381) This sequence of events involving processes and individual views which describes a possible direction in which the system may move is referred to as a *history*.

3. QSIM

Qualitative simulation (QSIM) was developed by Kuipers in 1985 (Iwasaki, 1989). QSIM takes a device, functions that describe the behavior of that device, and initial state facts and produces future states into which the device may transition based on the given information. "Given an initial qualitative state for each function, QSIM first generates all possible successor states for each function."(Iwasaki, 1989, p. 382) QSIM conducts a breadth-first generation of potential future states, filtering out those that are either redundant or inconsistent with the given facts.

The qualitative state of a function (behavior) is defined as a pair consisting of a qualitative value of the function and its direction of change (essentially the first derivative of the function with respect to time). Once a layer of possible future states of a device is generated, qualitative constraints can discriminate between valid and invalid states by restricting the range of qualitative values of a function and by limiting or forcing direction of change (Iwasaki, 1989, p. 386). Qualitative constraints are predicates which express some type of qualitative relationship among functions and can thus act as mechanisms to exclude impossible or contradictory states. Similarly, the qualitative constraint predicates act to eliminate illogical directions of change. Consider the constraint predicate ADD(f,g,h)

where f(t) + g(t) = h(t). Figure 3.1 (Iwasaki, 1989, p. 387) lists the possible valid directions of change and excludes those which would invalidate the predicate:

f / g	inc	std	dec
inc	inc	inc	any
std	inc	std	dec
dec	any	dec	dec

Figure 3.1

The use of qualitative constraints applied to both elements of the qualitative state pair has the potential to reduce substantially the list of future states for a device.

4. WQMS

World Qualitative Modeling System (WQMS) was developed in 1990 by Gaglio, Giacomini, Ponassi, and Ruggiero (Gaglio, et al., 1990). WQMS (1) models its domain using Forbus' QPT principles, (2) provides an interface for the user to input values and write results to a file, (3) provides a shell from which various active system views are processed, and (4) uses and Envision (ENV) simulator as well as QSIM simulator to move through the network of possible system states. The difference between the two is that ENV implements a depth-first search while QSIM uses a breadth-first search. Thus, ENV sacrifices the thorough examination of successive states provided by QSIM, but does not

get bogged down computationally when used for complex systems. The user is given the option of choosing between the two simulators at the beginning of a session.

WQMS was written in the production language OPS5 for the following reasons: First, production rules have the same conceptual structure as processes and individual views. Second, the inference engine of OPS5 can be used. Third, the efficiency of implementation is increased through the use of a rete matching algorithm, which provides several views or processes which satisfy the same conditions. "An important feature of OPS5 is its efficiency, due to the way in which rules are grouped in the conflict set by the rete match algorithm." (Gaglio, et al., 1990, p. 42) The rete match algorithm matches database elements against rules rather than rules against elements, making it easier to keep track of which rules apply and which do not when a database element is changed.

Individual views "represent a static situation in the evolution of a phenomenon." (Gaglio, et al., 1990, p. 43) In other words, they represent the things which are true when the system is in a particular state. Views and processes appear to be easily written in OPS5 and can also be coded somewhat generically. For example, the view which describes the heating process for water can also describe the heating process for steam.

Working memory initially contains the facts input by the user. Since each individual view corresponds to a rule, if the facts in the working memory support all the criteria of a view, one is created by the firing of a rule. Likewise, processes are activated when all the conditions for the process are true.

Two biological systems, cell growth and blood glucose dynamics, have been modeled using WQMS.

C. MODEL-BASED REASONING IN EDUCATION AND TRAINING

1. Model Development

A fundamental problem for students beginning the study of physics or advanced applied mathematics is a lack of conceptualization abilities and an unhealthy reliance on formulaic solutions. Research by White and Frederiksen (White and Frederiksen, 1989) contends that since traditional teaching relies on the use of quantitative laws in problem solving, and algebraic reasoning is substituted for underlying causal effects, there is a lack of connection between a student's *instinctive* notions of causality and the quantitative reasoning employed by textbooks and instructors.

White and Frederiksen employ the concept of an *articulate microworld* which combines qualitative modeling of electrical circuit behavior within the framework of an intelligent tutoring system (White and Frederiksen, 1989, p. 85). It is the primary vehicle in solving problems where the student is required to formulate mental models to understand domain phenomena and to solve problems. Models of system behavior progress from broadly qualitative and analogous to quantitative based on the student's progress and success in mastering the concepts and system generated test problems of lower level models.

White and Frederiksen discuss two distinctive types of qualitative models: *phenomenological* models and *reductionist* models. Their phenomenological models create systems that reason about gross circuit behavior. The first of these models adopted a "device-centered" view of causality (see ENVISION). System changes are reasoned from the perspective of individual components. The major drawback of the device-centered model is that it does not readily and intuitively model important laws that govern circuit behavior (such as Ohm's Law or Kirchoff's Current and Voltage Laws). Since the system reasons in a component-by-component fashion, the overall electrical process is not clearly

seen. This is not necessarily a bad thing when this model is viewed from the perspective of a new student thoroughly unfamiliar with electrical theory. However, since the spirit of the articulate microworld is to have different models for different levels of comprehension, other model methodologies must be used.

A "process-centered" phenomenological model was developed to describe the effects that system changes have on the system as a whole. A change in state of a component initiates a process in which all other devices within the affected part of the system ask whether or not they are affected by the change. If so, they alter their states appropriately. While the process-centered model shifts the focus from the component to the system as a whole, it still only describes (as opposed to *explains*) physical laws.

Reductionist models attempt to explain how changes is a system occur. These models split a system or component up into small, easy to understand parts and analyze the effects of a system change to the relationships between the parts. By reducing a device to smaller and smaller components, a more continuous cause/effect "chain" can be developed to explain why laws work as they do. Of course, care must be taken in deciding how far to reduce a device; the advantages of qualitative explanations lie in their relative simplicity.

Phenomenological, reductionist, and quantitative models work together in what White and Frederiksen describe as model evolution. "The evolution of knowledge can be captured as a progression of increasingly sophisticated causal models that are qualitative early on but that can later be mapped into quantitative models as students' understanding progresses."(White and Frederiksen, 1989, p. 94). They appear to fit very well with the notion of an adaptive cognitive model in the intelligent tutoring system field because they can be ordered into a hierarchy resembling how human beings learn things.

2. STEAMER

The STEAMER project was initiated in 1979 and developed through 1984 by Hollan in collaboration with several others, principally Hutchins and Weitzman (Hollan, et al., 1984). The domain of STEAMER is a Navy steam propulsion plant and its goal is to explore the use of artificial intelligence software and hardware in computer aided instruction (CAI). It is written in LISP.

Central to the development of STEAMER is the idea of *mental models*, the models people use to think about complex systems. "Without richer and more detailed understandings of the nature of these models, instructional applications will be severely limited." (Hollan, et. al., 1984, p. 15) Graphical interface is very important because the variations of how system interactions are presented are also variations on the level and direction of instruction. STEAMER presents an interactive, inspectable simulation; the user is permitted and encouraged to explore and inspect how system functions perform. "Interactive inspectable simulations have the potential of being major mechanisms for supporting the development of understandings of *process*." (Hollan, et. al., 1984, p. 15).

The domain area was chosen because the designers had access to a detailed mathematical simulation model of a 1200 psi steam plant. It is not strictly a qualitative model, but a quantitative one which presents information qualitatively. Another not insignificant factor in domain choice was "the potential for adequate research funding." (Hollan, et. al., 1984, p. 17) Moreover, a simulation restricted to user interaction via a keyboard, monitor and other peripherals is more cost effective than alternative plant simulations, which may cost as much as $7 million (Hollan, et. al., 1984, p. 17).

STEAMER uses dynamic interactive graphics as display engines which can represent the plant as the user would see it onboard a ship or as an expert would explain certain complex relationships (such as changes in a pneumatic control loop). The user has

the ability to alter plant parameters either by manipulating the devices that affect those parameters or by changing the parameters directly. Either approach allows the user to observe the effect that his or her change has on the plant.

A graphical editor allows a user to put "pieces" of a plant together. As these pieces are connected, LISP code is created for the new system. "In a number of tryouts of STEAMER in Navy schools, we have found that a short period of training is all that is required for instructors to begin to use the editor productively." (Hollan, et. al., 1984, p. 23).

The project developers contend that the difference between STEAMER and other AI efforts centers around its deep commitment to graphics in both the presentation and the editing. That may have been true in 1984; however, it needs to be reexamined today. The developers also contend that STEAMER-like systems provide a "qualitatively different and superior form of training." (Hollan, et. al., 1984, p. 26) It does employ model-based reasoning, but its foundations are quantitative. The display engines provide reasonable methods of extracting qualitative information from the quantitative data.

3. Intelligent Maintenance Training System (IMTS)

The Intelligent Maintenance Training System was developed by the Behavioral Technology Laboratories at the University of Southern California, funded in part by the Office of Naval Research (Towne, et al., 1990). It is an interactive graphical simulation that allows the user to build a system using a sort of graphical tool box. The user can then specify behavioral rules for each component in the new system. IMTS is implemented in Lisp.

IMTS simulations are built from generic objects contained in an object library. There are currently about 150 objects in the library. *Scenes*, which are screen-sized subsections of the simulation, are built from objects using the *screen editor*. When objects

are connected, basic rules regarding the interconnection are automatically generated. Generic objects come pre-coded with behavioral rules indexed by the possible states for the object. Each state has certain conditions which must be true for the state to be true, and certain effects which happen as a result of being in that state.

A user can create objects using a generic object authoring utility. After the object is drawn in its various states using conventional computer graphics, a rule definition must be specified. As in the generic objects, the rules must spell out conditions for each state and the effects each state propagates. Rule definitions can only be in terms of object input and output points (called *ports*) for both generic and created objects.

An important note is that scenes will not always act the way they are supposed to when first constructed. Common errors include unconnected ports in generic objects and inaccurate or incomplete rule definitions for created objects. IMTS is a simulation authoring system; it cannot incorporate first principles for specific systems (e.g., IMTS cannot assume that the output of Pump 'A' is the input for Valve 'B' because 'A' and 'B' may not exist together in any random system) (Towne, et. al., 1990, p. 38).

Simulations of varying levels of complexity can be created in IMTS. Simple models that mask system details can be developed for novices and more detailed (and more accurate) scenes can hone skills of advanced students. IMTS has never claimed to be a useful diagnostic authoring system. However, since the accuracy of any system is entirely dependent on the system author, there is no reason why a highly detailed and accurate model cannot be used (in conjunction with diagnostic code) to help pinpoint problem areas.

As of April 1990 there have been 11 different simulations built in IMTS. One very large one called Bladefold represents the mechanical, electrical and hydraulic processes for folding a helicopter's main rotor blades. It consists of 14 scenes containing 300+ specific objects and has provided a realistic training environment for advanced students (Towne, et. al., 1990, p. 40).

The major drawback of IMTS is its implementation. In order to reduce the most complex simulation in Bladefold from four minutes to 15 seconds, IMTS was recoded using machine-dependent direct-addressing techniques. As a result, the current version of IMTS can only run in the Xerox Interlisp Environment. Effective training in IMTS, then, can only be accomplished with heavy investment in a machine that is fast becoming archaic.

4. GTS

Generic Training System (GTS) was developed by Inui, Miyasaka, Kawamura, and Bourne (Inui, et al., 1989). Its goal is to effectively use artificial intelligence technology and qualitative reasoning techniques to build an individualized intelligent tutoring system. (Inui, et. al., 1989, p. 59) It is written in a variety of languages: Franz Lisp, OPS5, PEARL (Package for Efficient Access to Representations in Lisp) and Flavors. GTS combines knowledge representation schemes used in heuristic (rule-based) systems and qualitative models to offer a more robust training platform than traditional computer aided instruction systems.

Knowledge representation in qualitative simulation is implemented using the object-oriented features of the Flavors package. Constraints among objects are encoded as methods and are imbedded in the object descriptions (Inui, et. al., 1989, p. 63). Since the GTS project model is a power distribution system (PDS) containing many individual components, the device-centered approach proposed by de Kleer and Brown in ENVISION (called Device-Centered Ontology in GTS) was used as the model design strategy.

Objects are fitted into a hierarchy which uses the inheritance principles of Flavors. All objects in the PDS domain are subclasses of the basic switch. The design of this hierarchy, however, is not top-down but bottom-up. Each device is identified (such as transformers or disconnect switches) and common attributes of devices are used to create

or abstract superclasses. The abstraction continues until an indecomposable superclass, the switch, is formed. It is the most abstract object and contains all the "common" feature of all subclasses (Inui, et. al., 1989, p. 63).

The qualitative model is integrated into the ITS framework through the Model Based Tutor. It was originally thought that a student would need textual based instruction (from the Text Based Tutor layer) before model-based instruction could be used. However, system tests showed that the graphic displays and other model-based features variably mixed with text presentation offered the best results. The mix can vary from primarily text-based for entry level students to primarily model-based to offer more insightful explanations of difficult concepts for advanced students.

GTS is generic enough in principle to be used in a wide variety of intelligent tutoring domains. Since it relies heavily on model-based reasoning concepts, the domain should be one which is adaptive to those concepts. The power distribution prototype that developed as GTS developed has been expanded into a Power Distribution Training System currently in use at the Osaka Gas Training Center.

D. MODEL-BASED REASONING IN DIAGNOSTICS

1. ODS

Ontological Diagnostic System (ODS), written in LISP in 1989 by Gallanti, Stefanini, and Tomada (Gallanti, et al., 1989) relies on knowledge of formal design principles and an understanding of physical laws behind system operation to diagnose malfunctions. Like most model-based systems, the goal of ODS is to provide a deep knowledge network instead of a shallow knowledge base found in rule-based expert systems. However, unlike other model-based systems, ODS does not determine faults by failing likely components, allowing their new values to propagate through the model and then compare the new model values with the observed system values. Instead, ODS uses

models of the faulty behavior of devices to determine faults. The claim of ODS designers is that using these faulty models reduces the complexity of fault diagnosis, thereby making the whole process more effective and practical (Gallanti, et al., 1989, p. 143).

When a measurement value from an actual system sensor fails to meet established constraint values, the ODS diagnostic process is initiated. Discrimination among all possible causes of the malfunction is effected by propagating the corresponding malfunction models through the qualitative constraints and comparing the computed results with observed values.

The center of the diagnostic process is an algorithm which uses qualitative constraints as benchmarks with which it compares faulty-model generated values. The diagnostic machine considers the system input variables, design parameters affected by the malfunction, and other non-affected design parameters.

ODS designers claim superior performance when compared to an expert system based on empiric knowledge of the same domain (in the test case, a steam condenser). The better performance is due, in part, to the fact that the rule based system relied only on knowledge available in the plant control room while ODS fault models utilized parameters which, while not directly measurable, can affect plant operation (such as head loss through a section of piping). The intent of the ODS designers was to show that plant operators (who rely solely on observable measurement values) have not developed diagnostic skills tying unobservable measurements to observed ones (Gallanti, et al., 1989, p. 147). No surprise there.

ODS typically performed fault diagnosis in tenths of minutes on a Symbolics 3640 machine with 4 megabytes of main memory (Gallanti, et al., 1989, p. 148). No comparison was made between ODS and *any* other model-based fault diagnostic system. Such a comparison would have been a better gauge of performance than that with the rule-based (empiric knowledge) system.

The biggest stumbling block for ODS is its assumption that all possible component malfunctions can be modeled. "Fault and/or malfunction models are usually identified for most subsystems of industrial plants." (Gallanti, et al., 1989, p. 145) This may be true for simple devices or subsystems such as a light switch or even an electrical pump, but becomes exponentially more difficult for a device as complex as a propulsion boiler. Boiler malfunctions can be modelled efficiently only by breaking the boiler down into component parts simple enough for an exhaustive set of malfunction models to be developed. By that point, propagation of faulty models would be virtually the same as propagation of component values in traditional model-based reasoning systems.

2. Hoist

Hoist is a causal reasoning expert system based on qualitative physics. It was developed by Whitehead and Roach in 1990 (Whitehead and Roach, 1990). Hoist's domain is fault diagnosis in the lower hoist of the Mark 45 Naval gun turret. It reasons about machine failures from a functional model of the device, and is thus a model-based reasoning system.

The Hoist developers highlight three major shortcomings of rule-based expert systems in fault diagnosis. First, they cannot handle unanticipated faults. These faults are elsewhere referred to as "non-intuitive anomalies." (Fulton and Pepe, 1990, p. 55) Second, the development of a rule-based expert system is time consuming and there will, therefore, be a lag between the implementation of an actual real-world system and its corresponding expert system. Third, updates and design modifications may lead to incorrect or incomplete conclusions in the rule-based system. Hoist was designed for troubleshooting.

> "A repair expert might have some general rules for isolating faults, but s/he does not follow these rules exclusively, as shallow reasoning expert systems would imply. Instead, s/he understands the purpose of the machine and knows its expected behavior." (Whitehead and Roach, 1990, p. 109)

Unlike some other model-based expert systems, Hoist's first principles are cause-effect relationships between actual pieces of hardware and do not attempt to describe actions in terms of qualitative versions of physical laws. Since it models the gun turret at a high level of abstraction, some parts of the system cannot be accurately described. However, Hoist designers feel that the level of abstraction is generally sufficient enough to give a true rendering of the system as a whole.

Hoist is implemented in a language called WIF (What IF), which is based on counterfactual logic. WIF takes a "what if" introduced by the user and assumes it will contradict known facts. The model then generates all known worlds (states) which could exist is the counterfactual clause were true. This is ideal for troubleshooting because instead of matching symptoms to some set of rules, diagnosis starts by introducing suspected fault conditions and ascertaining whether or not the fault state can be reached given the "truth" of the suspected fault.

Hoist runs into combinatorially explosive situations when it is tasked to isolate multiple faults. However, the designers claim that heuristic searches through the "fault space" can reduce the effect of the explosion. (Whitehead and Roach, 1990, p. 116)

3. Mathematical Models and Uncertainty Theory

In the late 1980's, Lutcha and Zejda developed a fault diagnosis system for chemical processing units based on mathematical models (Lutcha and Zejda, 1990). Chemical production plants experience large financial losses when abnormal conditions force a shutdown. Moreover, the potential for loss of human life and the release of toxic chemicals is greatly increased. Development of a rule-based expert system was not considered well suited for chemical processing units because the operator expertise was deemed either inadequate or non-existent in many new and modified chemical processes.

Lutcha and Zejda proposed that all chemical processes, even the most complex, could be broken down into smaller, easier handled subsystems. These subsystems could be sufficiently described by mathematical equations founded on the principles of energy and material conservation (Lutcha and Zejda, 1990, p. 32). These mathematical equations, called *governing equations,* are expressed in terms of measurable variables, such as sensors. Sensor values are inferred to be discrete, and a set structure is used to keep all possible sensor fault states. Governing equations common to more than one subsystem provide the articulation points which link subsystems together into the original chemical process.

Since processes are broken down into subsystems, each of which is described by only a few governing equations, Boolean logic can be used to determine which sensor will fail given any other values in the mathematical models. These "other values" are actual measurement values from the processing unit. Problems arise when measurement noise causes the diagnosis to fluctuate between two or more faults. Lutcha and Zejda's solution to this problem is to introduce a certain level of belief of failure to each sensor for each discrete level of failure using Shafer-Dempster probability mass distributions (p.m.d). Take the p.m.d. for one governing equation (Lutcha and Zejda, 1990, p. 34):

$$m1\{H_1^+, H_1^0, H_1^-, F\} = \{0.8, 0.2, 0.0, 0.0\} \qquad \text{(eq 3.2)}$$

The certainty of some sensor state *F* being true is 0.8 if the result of the governing equation is higher than normal or 0.2 if the result of the governing equation is normal. The latter assumes that another sensor state may be counterbalancing *F* and further information (i.e., from other governing equations) is necessary to pinpoint the fault. So, while straight Boolean reduction results in only one fault (which may fluctuate with

fluctuating plant parameters), uncertainty theory assumes all faults are present and assigns a degree of belief (or weight) to each one (Lutcha and Zejda, 1990, p. 35).

Lutcha and Zejda programmed this system using Turbo-Pascal because of its ease in handling data structures and real number types. They are currently working on exploiting both shallow and deep knowledge in for an even more robust expert system.

IV. THE BOILER SYSTEM

Model-based reasoning is ideally suited for developing an expert system for a steam generation plant. As with any model, the better understood the system to be modeled, the more complete and robust that model will be. To that end, this chapter will focus first on the subcomponents that comprise the boiler system and how they work together. The second part of the chapter will examine what casualty conditions may arise and what their causes are.

A. BOILER FUNDAMENTALS

A 1200 psi steam plant is a complex arrangements of valves, piping, and components whose proper operation and interaction are critical to the operation of steam powered Naval vessels. The heart of the plant is the 1200 psi D-type boiler.

> "Boilers use thermal energy obtained from the combustion of fuel oil to change feedwater into superheated steam for use in the operation of the main engine, the turbogenerators and the main feed pumps. A portion of the superheated steam is desuperheated for use in the operation of auxiliary equipment." (Propulsion Plant Manual, 1978, p. 1-1)

It should be noted that although BoilerModel was developed based on the boiler and plant configuration of the FF-1052/1078 platform, the implementation is strictly qualitative (e.g., values of "LIFT_SAFE_HI" instead of "1385 psig" for lifting safety valves). Thus, the same model (with minor piping configuration changes) can be used for any Navy propulsion boiler.

1. Boiler Parts

The boiler components covered in this section are illustrated schematically in Figures 4.1 and 4.2.

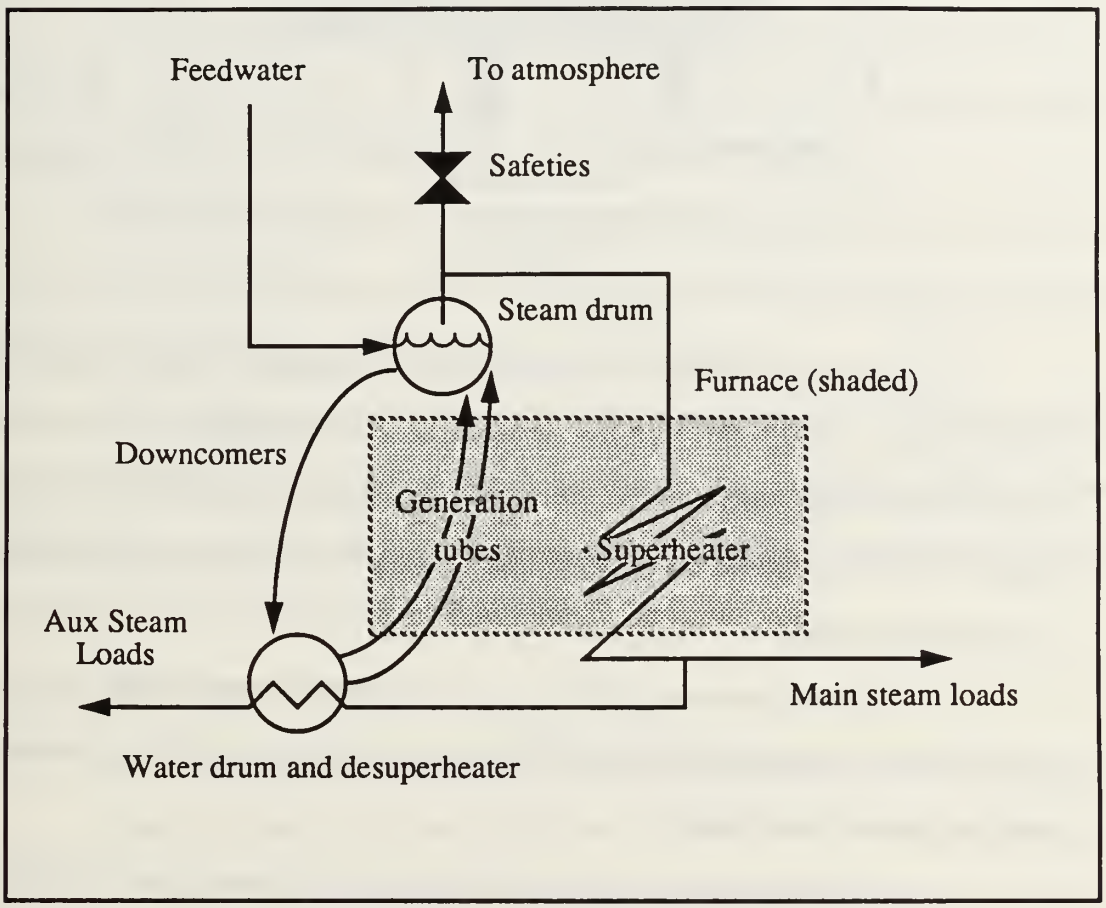


Figure 4.1

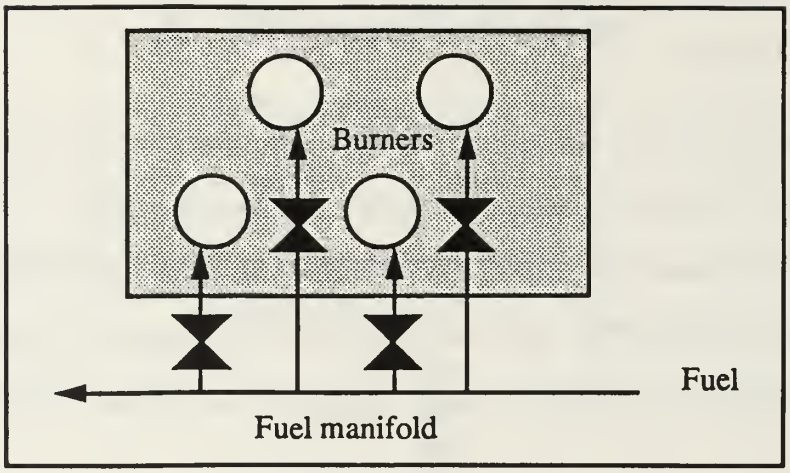


Figure 4.2

The *steam drum* is the largest pressure component of the boiler (Propulsion Plant Manual, 1978, p.1-20). It is physically located at the top of the structure. The steam drum is separated into two sections. The bottom half receives feedwater from the main feed pumps, which are external to the boiler itself. The top half receives saturated steam from the generation tubes.

The *water drum* is another large vessel which acts as a header, directing boiler water up the various generating tubes. The water in this drum acts as an indirect cooling medium for the desuperheater.

The *downcomers* are pipes which connect the steam drum to the water drum. They reside in the air casing which separates the exterior boiler surface from the furnace wall. Since they do not come into contact with direct heat and suffer virtually no casualties, the downcomers were not explicitly modeled in this project.

The *generation tubes* provide a conduit for boiler water to travel from the water drum back to the steam drum. These tubes come into direct contact with combustion gases in the furnace and are the primary places where water becomes steam.

The *superheater* is a tube bundle which receives saturated steam (steam at its boiling temperature for a given pressure) which becomes superheated as combustion gases flow past the tubes.

The *desuperheater* is located inside the water drum. It is also a tube bundle and uses the surrounding water to reduce the temperature of some of the superheated steam.

The *fuel manifold*, located on the boiler front, consists of valves and piping for four burners. Atomized fuel is mixed with combustion air from the forced draft blowers (external to the boiler) in the burners and is sprayed into the furnace.

The *boiler furnace* is a brick and refractory enclosure through which the generation tubes and superheater pass. When the boiler is on-line, the furnace is the home of a very hot fireball of burning fuel (courtesy of the burners).

The *safety valves* are located on top of the steam drum. They are adjustable, spring-shut valves that lift to relieve excess drum pressure and reseat below lifting pressure. Without safeties, the steam drum would be subject to accidental rupture, and the boiler room would be a more dangerous place to work.

2. The Boiler in Action

Figure 4.3 is a schematic of the major fireroom systems and main steam loads.

The main feed pumps provide relatively cool water to the boiler steam drum. Because of the lower temperature of the feedwater (compared to the temperature of the steam-water mix in the generating tubes), it falls naturally down the downcomers to the

water drum. It should be noted here that very pure "feedwater" becomes "boiler water" in the steam drum, where chemicals are injected to maintain pH and phosphate levels.

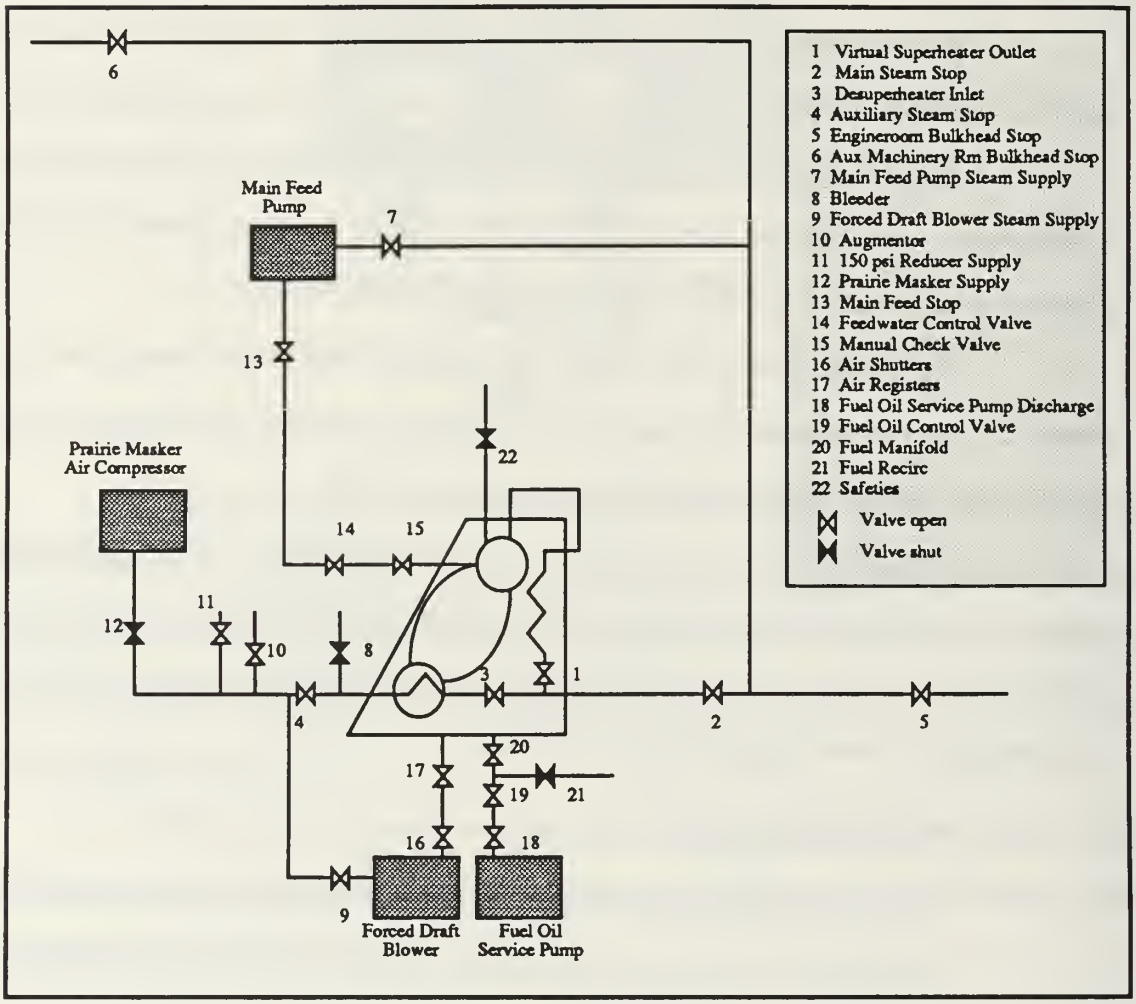


Figure 4.3

The boiler water is redirected by the water drum (which acts as a manifold) up the various generating banks. Natural circulation, which relies on the difference in densities between water drum water and steam drum steam, is the motive force behind this redirection. The water in the generating tubes proceeds to boil as it flows through the tubes to the top half of the steam drum. In a normally functioning boiler (safety valves shut),

there is only one place for the steam to go: to the superheater. The superheater has several passes, and as the steam flows through them, its temperature is raised above its boiling temperature (i.e., it is superheated). Superheated steam, also called *main steam*, is then available for the main engines, feed pumps, and tubine generators which require the high enthalpy from superheating.

Many steam systems, however, do not need steam at such a high temperature, so some of the superheated steam is directed to the desuperheater, which uses boiler water in the water drum to cool the steam some 300 degrees. From the desuperheater, the *auxiliary steam* is used for all other steam loads, including heating systems, laundry and scullery operation, atomization steam, and forced draft blowers.

Fuel is provided to the fuel manifold by the fuel oil service pump. It is atomized by steam and mixed with just the right amount of air to burn cleanly and efficiently. Combustion air is provided by turbine driven forced draft blowers via an air register assembly at the burner front.

B. BOILER CASUALTIES

Many abnormal conditions may face a boiler operator. Most, if left unchecked, will cascade into more severe casualties, resulting in equipment damage and/or loss of life. This section will examine some of the more common boiler casualties along with their symptoms, indications, and possible causes.

1. Fuel on Deck

Normally, all fuel supplied to the burner front is consumed by the fireball in the furnace. Occasionally, though, unburned fuel may leak from the atomizer to the furnace floor. The cause of this problem usually resides with the atomizer itself; it has not been tightened sufficiently or is partially clogged by water or sediment. Occasionally, fuel will drip on deck when the boiler is secured using the Fuel Oil Quick-Closing Valve. There are

three types of Fuel on Deck casualties, but only one, *Unburned Fuel on Deck with Fires Secured*, is currently incorporated in BoilerModel.

Fuel in this condition evaporates rapidly because of the residual heat in the furnace. The evaporating fuel will fog the boiler periscope mirror. A large amount of fuel vapors in the fire box presents a potentially explosive situation.

2. White Smoke

A precise mixture of fuel and air is required for clean combustion (i.e., no smoke and minimal soot build-up on tubes). When the mixture becomes unbalanced and there is too much air flow, a white smoke condition may result. White smoke will appear orange in color through the boiler periscope, and billows of whitish-grey smoke will come from the stacks topside. Additionally, superheater temperature may be lower than normal. In this condition, unburned fuel in an aerosol state (from the excess air) travels up the stack. Since the stack is not a clear, straight path up from the furnace, the aerosol may be diverted into eddies and may cool down enough for fuel to condense onto hot surfaces, resulting in a boiler explosion. Sixty seconds is considered the maximum time a white smoke condition may exist uncorrected. The likelihood of a boiler or stack explosion increases greatly after that point.

3. Black Smoke

When the fuel/air mixture becomes too fuel-rich, a black smoke condition will occur. This is indicated by a blacked-out periscope and large amounts of dark exhaust from the stacks. Although not as dangerous and time critical as a white smoke casualty, black smoke increases the amount of soot on tubes and along the stack wall, and increases the likelihood of a stack fire.

4. Low Water

Water is not only the working fluid in the boiler system; it is also the cooling medium for the generation tubes and the superheater. A low water condition will arise when there is greater flow out of the boiler than there is flow in. A malfunctioning Feedwater Control Valve is frequently the culprit in a low water casualty. The primary indicator for this casualty is an alarm which sounds when steam drum water level drops to -6 inches (i.e., six inches below normal water level). Before and after the alarm sounds, steam drum level can be tracked directly via a gauge glass on the steam drum. As water level continues to drop, tube ends and eventually the tubes themselves are deprived of their cooling fluid and become subject to deformation to the point of rupture.

5. High Water

More water supplied to the boiler than steam flow out will result in a high water casualty. High water is not as catastrophic to the boiler as low water, although it may result in the carryover of water (and with it, chlorine ions) into the superheater. Since the superheater is made from stainless steel, it is subject to *chloride stress corrosion* at high temperatures. The worst effects of a high water casualty are seen in turbine driven equipment. A slug of water carried out of the boiler can shatter the blading of a turbine rotating thousands of revolutions per minute. High water is indicated by an alarm when steam drum water level reaches +7 inches.

6. Ruptured Tube

A ruptured tube occurs when there is inadequate heat transfer from tube surface to cooling medium. A frequent cause for a ruptured tube is a low water condition, but other causes include improper boiler water chemistry and fouled watersides. A ruptured tube is a catastrophic casualty; the boiler cannot be immediately restored. Moreover, ruptured tubes often inflict collateral damage on other nearby tubes and on the furnace brickwork.

A loud hissing, fogged periscope, and sudden drop in steam drum pressure are indicators of a ruptured tube condition.

V. AN ADA IMPLEMENTATION

From its earliest conception, BoilerModel was meant to be an Ada project. Several factors, including speed, maintainability, and portability contributed to this decision; however, the main consideration was the Department of Defense's embracement of Ada as a *lingua franca* for future programming applications. This chapter will first examine the scope of the BoilerModel implementation. Next, it will assess the value of Ada as an artificial intelligence tool. Finally, the model implementation and results from a test run will be critically reviewed.

A. SCOPE OF THE MODEL

The original plan for BoilerModel was to write and implement it on an IBM-type PC using Meridian Software's AdaZ (later OpenAda) compiler. An early version of BoilerModel was written and did run with AdaZ; however, the variable stack used by the compiler later proved to be inadequate for the number of global variables (and the size of the data structures in which these variables were instantiated) in the current version of the model. With virtually no changes to existing code, the model was transferred to a Sun SPARCstation and the Verdix Ada compiler. The number and size of global variables did not adversely affect that compiler.

BoilerModel models a somewhat simplified 1200 psi D-type boiler, along with valve and piping systems to and from major loads and supporting auxiliary equipment. Although all propulsion boilers operate the same in principle, BoilerModel's architecture comes from the FF 1052/1078 class platform. For the purpose of this implementation, boiler steam loads are assumed to be "receive-ready" and boiler auxiliaries are assumed to be "supply-ready." This simply means that if, for example, the boiler is on-line and an open path to the

Engineroom for main steam exists, then the steam will be used in the Engineroom (even though, at present, there is no such end-user in the model). Likewise, if there is an open piping path from the Fuel Oil Service Pump, then fuel will flow to the boiler regardless of the fire status of the furnace.

Assumptions like these have their problems. For example, the Main Feed Pumps on a frigate are steam driven. However, since they have not been fully modeled here, they will still operate when steam flow from the boiler is secured. The "receive-ready" and "supply-ready" assumptions should be viewed as temporarily undeveloped components in a larger propulsion plant model. They currently serve as a test harness for the boiler.

The Automatic Boiler Control (ABC) systems were not included in this model; they are complex enough to comprise a separate project. Since they are measurable, interacting physical systems, they can also be implemented in a model-based reasoning system to work with BoilerModel.

Finally, a valve which does not exist on the real-world boiler was included in this model. The Virtual Superheater Outlet was added so the user could observe the effects of stopping all steam flow from the boiler. A later version of BoilerModel should contain a more versatile user interface which would allow the user to change more than one valve status or characteristic per scenario. That versatility is currently lacking.

B. ADA IN ARTIFICIAL INTELLIGENCE

The typical benchmark in artificial intelligence technology is "adequacy"-- does the system provide acceptably correct answers or diagnoses in an acceptable amount of time or detail? Programs have generally been prototyped in one of the standard AI languages, such as LISP, and once developed, translated into a more efficient language (e.g., C or Pascal). (Baker, 1987, p. 39)

Ada provides an alternate solution. Its rich data types, capability for multitasking, and strong typing requirements are some of the reasons Ada can and should be used from initial program development through implementation of the final product.

> "Ada is a language that directly embodies many modern software engineering principles and is therefore an excellent vehicle with which to express programming solutions. Ada not only encourages the use of good design and programming practices but . . . it can actually enforce such practices." (Booch, 1987, p. 4)

This section will focus on the characteristics of Ada which can make it a preeminent artificial intelligence tool.

1. Data Types and Strong Typing

Virtually anything that can be modeled can be modeled using Ada types. There are four categories of Ada types: scalar types, composite types, access types, and private types (Booch, 1987, p. 104). *Scalar types* describe values consisting of a single component. They include integers, reals, and enumeration types (e.g., characters or Booleans). *Composite types* deal with objects that are logically composed of different components, such as array and record types. *Access types* are dynamically created and destroyed objects that may, by reference, belong to more than one other object. Access types are known as pointers in C and Pascal. *Private types* allow the developer to explicitly define the structure and value of a type and provide for the user only those operations necessary for the manipulation of object attributes consistent with design. The integrity of an object declared as a private type is thus ensured (Bennett, 1991, p. 33).

Ada is a *strongly typed* language. This means that variables of a certain type can assume only those values which are appropriate for that type. Additionally, operations performed on a variable must be predefined in the type definition for that variable type. For example, suppose the enumeration type MEASUREMENT_VALUE was declared (with HIGH, LOW, and RESULT instantiations of that type) and a programmer wrote the following line of code:

```
RESULT:= HIGH + LOW;
```

The compiler would reject the program because enumeration types have no defined addition operator. Other languages, such as Lisp, would allow such a line of code to compile, but would yield some unpredictable result if that line were invoked. Ada is thus a more difficult language in which to write a *compilable* program, but that program will yield correct and consistent results when run. The advantage here lies in the fact that compiler errors are usually identified by line; debugging is a matter of correcting that error (which may not be a trivial exercise). Run-time errors do not explicitly identify which section of the program was at fault and can lead to hours of line-by-line examination of code.

Ada data structures range from the common (linked lists) to the reusable (generic program units). They can be used as abstractions of real-world objects or as logical groupings for related types. Their versatility is limited only by the typing constraints of their component types and the memory constraints of the compiler. Because they can be precisely tailored for a given application, they are ideal for use in model-based reasoning systems which depend on accurate component definition and inter-component connectivity.

2. Multitasking

An Ada *task* is one of the primary program units of the language (along with subprograms and packages). Tasks may or may not communicate with other program units, and may be assigned different priorities. All tasks, however, have one thing in common: they are designed for concurrent processing. A machine with three processors can run three tasks from the same program at the same time. A machine with only one processor must rely on an implementation-dependent scheduler which may not necessarily be "fair" (Booch, 1987, p. 282). Moreover, most Ada compilers contain some sort of time-slicing

algorithm that can be used to simulate concurrent processing (the state of the task when its time slice has expired is saved and it is at that point that the task will resume when it gets a new slice).

There are two general classifications of tasks: *actor tasks* and *server tasks*. Actor tasks are not called (or "entered") by any other tasks or program units and are thus continuously active. They may, in turn, activate other tasks with visible entry points. Server tasks have entry points but cannot activate any other tasks. Tasks which are hybrids of these two types can be created. (Booch, 1987, p. 283)

Care must be taken to ensure that tasks do not inadvertently corrupt variables shared with other tasks. For example, Task A uses the value of a variable x in computations. It must be guaranteed that if Task A is preempted by a higher priority task or its time slice has expired before the end of the task, then no other task will corrupt x before Task A can complete its computations.

In a steam generation plant, several events occur simultaneously. Steam flows through piping systems at the same time as fuel is supplied to the boiler at the same time as feedwater is pumped into the steam drum. Ada tasks are outstanding tools for modeling the cause-effect relationships in such a system. For example, when fires go out in a real-world boiler, steam flow out of the generation tubes is immediately reduced. Two tasks, one which concerns itself with boiler fires management and another which monitors steam flow through boiler tubes could run independently yet share a common variable: boiler fire status (changeable only by the fires manager). Now, instead of having the disjointed nest of *if* and *case* statements and an unrealistic sequence of events common in a sequential processing system, one can realistically model events which occur concurrently.

3. Portability and Speed

A machine-dependent artificial intelligence application is useful only as long as the particular machine is available, affordable, and multi-purpose. Similarly, programs written in languages lacking a common standard are neither easily maintained nor readily integrated into other applications written in different dialects of the same language. Lisp and C are languages in which portability can be a problem. C is generally portable, but libraries vary from implementation to implementation. Since C is a language of functions, this can be a difficult problem to overcome (Baker, 1987, p. 40). Lisp has traditionally been very nonportable (Baker, 1987, p. 40), although efforts have been made to standardize Common Lisp. Ada is currently the most portable, "although at present this portability is limited by the availability of Ada compilers and support environments." (Baker, 1987, p. 40) Since an Ada compiler may only be authorized for use in DoD applications if it conforms to the ANSI/MIL-STD-1815A requirements promulgated by the Department of Defense, it can be a time and money consuming proposition to build a compiler. There are, however, several more on the market since Baker (1987), and they are affordable. Ada's portability was put to the test during the development of BoilerModel. Code for an early version of the project that had compiled and was successfully running on an 80286 machine was transferred in ASCII format to a UNIX based Sun system. No changes to the code were needed for it to compile and run on the new system.

Ada generates code which, while probably somewhat slower than C code, is markedly faster than Lisp. This comes as no surprise; "Lisp programs are great consumers of memory and often CPU time." (Baker, 1987, p. 39) One of the design considerations for Ada was real-time control (for use in embedded systems). To that end, one of the three goals established by the Ada language team was *efficiency*. "Any language construct whose implementation was unclear or required excessive machine resources was rejected."

(Booch, 1987, p. 54) Ada's speed would be of great advantage in real-time expert systems, such as autonomous vehicle control and robot sensor processing.

4. Readability and Maintainability

An argument can be made that Ada code is easier to read than Lisp code for most people raised on traditional programming (Baker, 1987, p. 43). Its English-like syntax (no *car*'s or *cdr*'s, thank you), minimal use of parentheses, and modular design certainly enhance its appeal. If the language is more readable, then it will probably be more maintainable. "Since Ada is a highly structured language, it is easy to maintain the original structure of the system while modifying pieces." (Booch, 1987, pp. 422-23) Of course, the bottom line as far as readability goes will probably be personal preference. Ada supporters claim that an Ada program can be understood easily and translated into other languages (Baker, 1987, pp. 39, 40). In fact, this claim is used to promote the general utility of the language. Can Lisp supporters make such a claim?

C. ADA vs. LISP -- A CASE-BASED COMPARISON

An early version of BoilerModel (hereafter referred to as ProtoBoiler to differentiate it from the final version) that did not incorporate Ada's tasking constructs was compared to the same program written in Lisp. The code for both these programs can be found in Appendix C. It should be noted here that the Lisp program was written as functionally as possible to ensure that the comparison fairly evaluated an Ada program against a Lisp program as they are conventionally written. Tables 5.1, 5.2, and 5.3 synopsize the results of the test. The Lisp code was written and run in the Allegro Common Lisp environment in both an uncompiled and a compiled version. The difference in speed is at the expense of storage (16.2 K vice 36.7 K). Since memory is no longer a consideration for all practical purposes, this trade-off is worthwhile. Additionally, both Lisp versions have time for "garbage collection" and "non-garbage collection" use of the CPU. Garbage collection

means that when usable memory is full, the processor stops what it is doing to recover unreferenced memory (Baker, 1987, p. 40). Although garbage collection does vary depending on system usage, it is a real time consumer which must be taken into consideration. The Ada compiler performs garbage collection only once, at compile time. All comparisons were made at the same time of day, with similar system loads.

ADA

		TIME (SEC)		
VALVE	**TRACE**	**USER**	**SYSTEM**	**TOTAL**
FOCV	Y	0.3	1.3	1.6
FOCV	N	0.0	0.1	0.1
MSS	Y	0.1	0.3	0.4
MSS	N	0.0	0.0	0.0
FEED STOP	Y	0.0	0.3	0.3
FEED STOP	N	0.0	0.1	0.1
TEST1*	Y	0.7	2.3	3.0
TEST1*	N	0.0	0.2	0.2

Storage: code 15799 bytes
executable 229376 bytes

* TEST1 closes FOCV, then DESUP-IN, then MSS

Table 5.1

UNCOMPILED LISP

VALVE	TRACE	NON-GC TIME (SEC) USER	NON-GC TIME (SEC) SYSTEM	GC TIME (SEC) USER	GC TIME (SEC) SYSTEM	TOTAL
FOCV	Y	9.3	2.9	0.9	0.9	14.0
FOCV	N	1.0	0.8	0.0	0.0	1.8
MSS	Y	3.0	4.7	0.0	0.0	7.7
MSS	N	0.8	0.7	0.0	0.0	1.5
FEED STOP	Y	2.9	4.6	0.0	0.0	7.5
FEED STOP	N	0.7	0.8	0.0	0.0	1.5
TEST1*	Y	17.6	5.9	1.7	1.1	26.3
TEST1*	N	1.7	0.6	0.0	0.0	2.3

Storage: code 16228 bytes

* TEST1 closes FOCV, then DESUP-IN, then MSS

Table 5.2

COMPILED LISP

VALVE	TRACE	NON-GC TIME (SEC) USER	NON-GC TIME (SEC) SYSTEM	GC TIME (SEC) USER	GC TIME (SEC) SYSTEM	TOTAL
FOCV	Y	3.7	2.3	0.0	0.0	6.0
FOCV	N	0.2	0.1	0.0	0.0	0.3
MSS	Y	1.1	0.6	0.0	0.0	1.7
MSS	N	0.2	0.0	0.0	0.0	0.2
FEED STOP	Y	1.0	0.6	0.0	0.0	1.6
FEED STOP	N	0.2	0.1	0.0	0.0	0.3
TEST1*	Y	7.5	4.1	0.0	0.0	11.6
TEST1*	N	0.6	0.2	0.0	0.0	0.8

Storage: code 36710 bytes

* TEST1 closes FOCV, then DESUP-IN, then MSS

Table 5.3

Four test cases were used in the comparison. The first three propagated changes when one valve was closed (Fuel Oil Control Valve in case 1, Main Steam Stop in case 2, and Main Feed Stop in case 3). The fourth case closed three valves (Fuel Oil Control Valve, Desuperheater Inlet, and then Main Steam Stop). The four cases represent the major systems integrated in ProtoBoiler. Each case was run with "trace" on and "trace" off. "Trace" enables the user to watch the propagation of values as they occur. With "trace" off, the user would only see the initial and final plant statuses.

The Ada program ran consistently faster than either Lisp version. "TEST1," which closes multiple valves, ran almost nine times faster than uncompiled Lisp and almost four times faster than the compiled Lisp version (all three with "trace" on). The Ada code required 15.8 K storage versus 16.2 K and 36.7 K for the uncompiled and compiled Lisp versions, respectively. The executable Ada program (which runs independently in the UNIX shell) required 229.4 K. The Lisp code requires the Allegro environment to run. Although, as previously asserted, memory is not a big concern in the test environment (Sun SPARCstation with UNIX operating system), the size of the executable code or all systems required to run the program may be a consideration for other machines. PC's running MS-DOS or PC-DOS may be limited to executable files less than 640 kilobytes.

D. DATA TYPES AND STRUCTURES IN BOILERMODEL

The code for BoilerModel can be found in Appendix A. The prominent data types used are simple enumeration types, arrays, and records. The main data structure used is the linked list. This section will examine in detail BoilerModel's two major records, VALVE and BOILER. In the course of this examination, all important data types and structures used in this implementation will be covered.

1. VALVE Record

Type VALVE is a record that contains all the information and parameters necessary for valve operation in BoilerModel. Each valve in the system is an instantiation of an access type, VALVE_PTR, which points to a valve record. The valves in BoilerModel are connected in a linked list structure. The fields of type VALVE will be discussed in this subsection.

a. VALVE_ID

VALVE_ID is an instantiation of a previously declared enumeration type, INDEX. It permits identification of an individual valve in the user trace function since the name of an access type instance cannot be output.

b. UPSTREAM and DOWNSTREAM

UPSTREAM and DOWNSTREAM are arrays of type VALVE_PTR. They contain pointers to the valves which are immediately upstream or downstream of each valve. Normally, there is only one valve in each of these arrays; however, some valves, such as MAIN_STEAM_STOP, act as distributors for several downstream systems or receivers from multiple sources and thus require several valves in one of the two arrays.

c. COUNTED

The list containing the valves is circularly linked to allow monitoring by a procedure that checks for flow in the system (to be discussed later). The Boolean COUNT is used to prevent endless cycling through the list when a single traversal is needed.

d. NEXT

NEXT is the VALVE_PTR that acts as the connector in the linked list of valves.

e. STATUS

The STATUS of a valve is either OPEN or SHUT. OPEN and SHUT are instances of the enumeration type MEASUREMENT_VALUE. PREV_STATUS is the status of a valve the *last* time the propagation task looked at it.

f. Pressure and Flow

Pressure and flow in and out of a valve are maintained as distinct fields. They are of type MEASUREMENT_VALUE and can take on the values of NONE, LOW, NORM, HIGH, and LIFT_SAFE_HI. Each input and output pressure and flow has a corresponding "PREV_" field which holds the value its counterpart held the last time the propagation task looked at it.

g. SYSTEM

Each valve belongs to one of four systems: STEAM_SYSTEM, AIR_SYSTEM, FUEL_SYSTEM, or FEED_SYSTEM. All systems are arrays of VALVE_PTR. Since it is possible that propagation of values can occur concurrently in two systems, this field allows the value of COUNTED to be reset in one system without affecting any other.

2. BOILER Record

Type BOILER is a record of other records. Its constituent members consist of STEAM_DRUM and WATER_DRUM (instances of type DRUM), SUPERHEATER, DESUPERHEATER, and GENERATION_TUBES (instances of type TUBE), and FURNACE (an instance of type BOILER_FURNACE). The field types (DRUM, TUBE, and BOILER_FURNACE) will be discussed in this subsection.

a. DRUM

Type DRUM is a record whose fields are all instantiations of type MEASUREMENT_VALUE. WATER_LEVEL may take on the values NONE, LOW_ALARM, LOW, NORM, HIGH, and HIGH_ALARM. PRESSURE, FLOW_IN, and FLOW_OUT are similar in nature to their counterparts in type VALVE. TEMP (temperature) can assume the values LOW, NORM, or HIGH. It has no meaning for the

water drum (where it is not measured in a real-world boiler), and is not shown on the boiler status display. There are "PREV_" values for all these fields, as in type VALVE. These values, however, are not used for propagation purposes, but are used in determining whether an updated boiler status needs to be displayed.

b. TUBE

Type TUBE is a record that contains values for input and output flow and temperature of a tube. There is a Boolean, RUPTURE, that is set when a tube is ruptured. There are "PREV_" values for all these fields. One final element of type TUBE is another Boolean, DIRECT_HEAT_CONTACT. The procedure that checks for a ruptured tube must first ascertain whether or not the tube in question comes into direct contact with flames or combustion gases. If it does not (as is the case of the desuperheater), a rupture due to inadequate flow through the tube will not occur.

c. BOILER_FURNACE

Type BOILER_FURNACE contains much of the data needed to correctly diagnose or propagate casualties. Real-world boiler operators rely heavily on information directly or indirectly observed from the furnace. The BOILER_FURNACE fields are detailed in this subsection.

(1) DECK_STATUS. DECK_STATUS indicates whether or not there is fuel on the deck of the furnace.

(2) FIRING_RATE. FIRING_RATE measures the intensity of the boiler fireball, and varies directly with fuel manifold input pressure. FIRING_RATE is one of the controllers of steam drum pressure. It may take on the values of NONE, LOW, NORM, and HIGH.

(3) EXPLOSION. EXPLOSION is a Boolean that indicates the presence or absence of a boiler explosion.

(4) PERISCOPE. The boiler periscope can give an indication of a white or black smoke condition or fuel on deck. PERISCOPE can assume the values of CLEAR, BLACK, ORANGE, or FOGGED.

(5) FIRES_LIT. FIRES_LIT is a Boolean indicating the status of boiler fires, and thus the boiler itself (a boiler is considered off-line when fires are extinguished).

(6) FIRE_APPEARANCE. Boiler fires may appear FAN_SHAPED (normal) or IRREGULAR. IRREGULAR fire appearance indicates a smoke condition, an air problem, or a fuel problem.

(7) Previous Values. The above fields have corresponding "PREV_" fields to facilitate boiler status display updates.

E. BOILERMODEL TASKS

Four tasks drive BoilerModel: PROPAGATE, STEAM_DRUM_MANAGER, FIRES_MANAGER, and TUBE_MANAGER. They are all actor tasks, each embedded in a *loop* statement. Thus, once activated, they continually perform updates and constraint propagation. Since they are all of the same priority, they are scheduled using an implementation-defined First-In, First-Out ready queue.

It should be noted here that the main procedure of any Ada program is an implicit task. Procedure MAIN in BoilerModel is no different. It is assigned a higher priority than any other task so it can perform boiler and plant initialization before propagation begins (unpredictable and erroneous results occurred when MAIN was assigned a priority equal to the other tasks). MAIN provides user interface by querying the user about what valve status

or characteristic to change and displaying boiler status when a boiler parameter has been changed.

This section will describe the function of the four explicit tasks that are the workhorses for BoilerModel. The structural relationships of program units can be found in Appendix D.

1. PROPAGATE

Task PROPAGATE takes a look at current and previous status, pressure, and flow for each valve in the linked list of plant valves. If a current and previous parameter of a valve do not match, it must mean that some value was propagated to that valve and must continue propagating until it reaches a sink (user) or a dead end. To do this, PROPAGATE calls procedure PROPAGATE_VALVE_VALUES.

PROPAGATE_VALVE_VALUES first determines whether the valve under consideration is open or shut. If it is open, then output pressure and flow are assigned the same values as input pressure and flow (much the same as in the simple valve and piping arrangement in Figure 2.2) If, on the other hand, the valve in question is shut, then output pressure and flow from that valve is reduced to nothing. Additionally, a determination must be made as to whether there is flow in the parent system with that valve shut. PROPAGATE_VALVE_VALUES calls procedure CHECK_FOR_FLOW to make that determination.

CHECK_FOR_FLOW recursively moves upstream one valve and then performs a depth-first search for an open path to a sink. If such a path is found, one of the input paramenters, FLOW (a Boolean), is set to *true*; otherwise, it remains *false*.

In the case that no path can be found from source to sink in a system, an overpressurization will occur. Procedure OVER_PRESSURE moves recursively upstream from the original valve, propagating HIGH output and input pressures. Task

PROPAGATE will pick up the lack of output pressure and flow from a shut valve and conduct normal propagation of values downstream.

Procedure PROPAGATE_VALVE_VALUES does two final things. First, it resets all "PREV_" values for the affected valve. Second, it assigns the output pressure and flow of the subject valve to each of the valves immediately downstream. Task PROPAGATE will pick up the changes to these new valves next time through its loop; thus, constraint propagation through a valve and piping system is effected.

2. STEAM_DRUM_MANAGER

Task STEAM_DRUM_MANAGER regulates water level in the boiler steam drum. To do this, it compares the flow into the steam drum with the flow out. The flow through the boiler tubing itself is controlled by another task, TUBE_MANAGER. If there is more flow into the boiler than flow out, water level will increase first to a HIGH condition and then to HIGH_ALARM (signalling a High Water casualty). If the flow out of the boiler is greater than the flow in, water level drops to LOW, and then to LOW_ALARM (a Low Water casualty). NORM water level is the equilibrium condition.

STEAM_DRUM_MANAGER also controls safety valve operation. If the input pressure to the VIRTUAL_SH_OUTLET is HIGH, then steam drum pressure will rise to LIFT_SAFE_HI and safety valves will OPEN. Safety valves are small in comparison to stop valves (such as the Main Steam Stop and Auxiliary Steam Stop) on a real-world boiler. Consequently, steam drum pressure cannot drop from LIFT_SAFE_HI to NORM instantly. To realistically simulate the slower drop in pressure, STEAM_DRUM_MANAGER calls procedure INCREMENTAL_DECREASE. INCREMENTAL_DECREASE drops steam drum pressure from LIFT_SAFE_HI to HIGH to NORM with delays at each drop. Safety valves will not shut until steam drum pressure is NORM because safety valves reseat at a lower pressure than they lift.

Delay statements, which have the effect of blocking and rescheduling a task, have been added at various points in the tasks. Since the routine to display boiler status is called from procedure MAIN (an implicit task), strategically placed *delays* permit a more comprehensive view of boiler changes.

3. FIRES_MANAGER

Task FIRES_MANAGER controls steam drum pressure by regulating firing rate based on fuel manifold output pressure. In a real-world boiler, an Automatic Combustion Control (ACC) system regulates fuel and air pressure (and therefore firing rate) to maintain normal steam drum pressure. The ACC system is one part of the Automatic Boiler Controls system which was not incorporated in BoilerModel. However, to mirror real-world boiler operations as closely as possible, boiler firing rate (more accurately, Fuel Oil Control Valve output pressure) changes only when steam drum pressure varies from NORM. When drum pressure is greater than NORM, firing rate decreases; when steam drum pressure drops below NORM, firing rate increases. When steam drum pressure reaches NORM, firing rate becomes NORM. In a real-world steam generation system, NORM depends on the boiler loads (NORM is greater when the ship travels at higher speeds, for example). So, even though the boiler firing rate may not change quantitatively during the transition from abnormal to normal steam drum pressure, it *does* change qualitatively to reflect the new normal fuel and air demand for the changed steam load.

The boiler furnace parameters are also controlled by FIRES_MANAGER via a set of constraints. The constraints constitute the necessary preconditions for furnace values to be other than normal. For example, if there is a path for fuel into the boiler and the fuel is being supplied and fires happen to be extinguished, then the furnace deck will have fuel on it and the periscope will be fogged. How the constraint values come to be is

of no concern to FIRES_MANAGER; only the cause-effect relationships across the furnace subcomponents is regulated. Hence, fuel pressure of NONE to the boiler can result from task PROPAGATE, while a fire appearance of NONE and LOW steam drum pressure are effected by task FIRES_MANAGER.

4. TUBE_MANAGER

Task TUBE_MANAGER controls the flow of water and steam through the boiler, from the Manual Check Valve output to the Main and Auxiliary Steam Stops. Figure 5.1 is a schematic representation of boiler flow. TUBE_MANAGER ensures connectivity by assigning values for component input flow based on the appropriate component output flow.

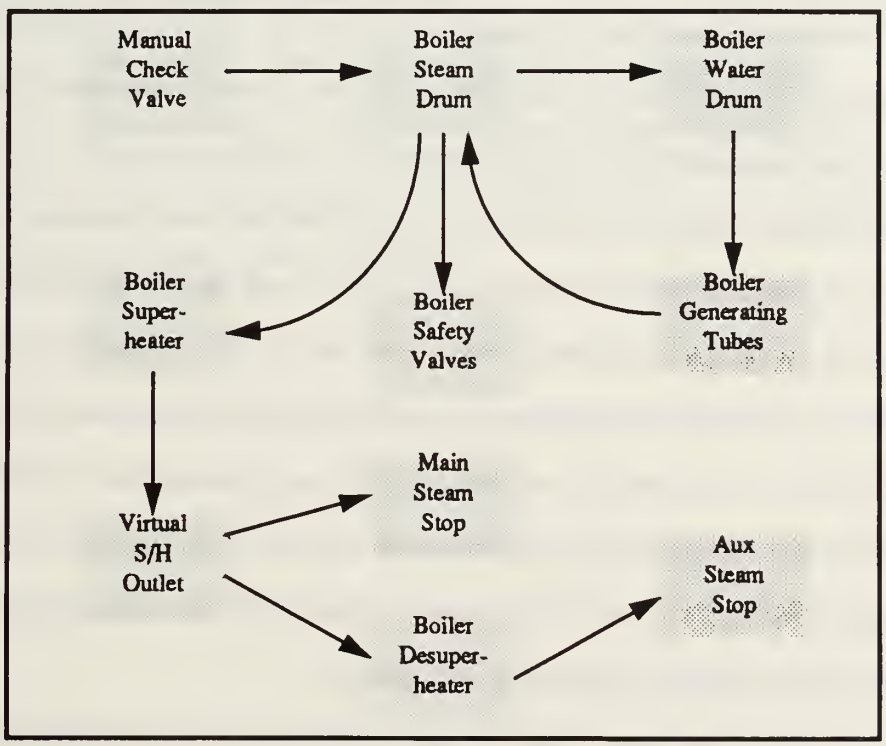


Figure 5.1

TUBE_MANAGER calls procedure RUPTURED_TUBE_CHECK for each of the boiler tube bundles. If the flow into a tube is less than the flow out for an extended period of time, then tube temperature will rise. If the tube is actively exposed to combustion gases (i.e., FIRES_LIT:= TRUE and DIRECT_HEAT_CONTACT:= TRUE), then a rupture will occur if flow is not normalized.

F. RESULTS

The results of a comprehensive test run of BoilerModel can be found in Appendix B. The user interface permits the user to choose between changing a valve *characteristic* or a valve *status*. The characteristics that can be changed are *input flow* and *input pressure*. Each can be changed to *none, low, norm,* or *high*. Altering a characteristic allows user control over what values will propagate and where propagation will start. A change in valve status more accurately mirrors how an operator can effect changes to the plant: by opening or closing a valve. Both methods of change were undertaken in the test run.

In all cases, the end results of propagation match the expected results in a real-world boiler system. This is a good thing, but could have been accomplished without difficulty using a rule-based expert system. The model-based nature of BoilerModel, however, permits the user to incrementally trace changes as they propagate through the system. Moreover, at any point in time the plant status is *relatively* accurate; events occur and are displayed in correct relation to other events. (e.g., propagation of low steam pressure through the main steam system can occur only after the user can *see* that something happened to change steam drum pressure). One of the test cases, shutting the Feedwater Control Valve, will be examined in this section.

When the Feedwater Control Valve (FWCV) is shut, two things happen that initiate the causal chain of events. First, since shutting the FWCV in BoilerModel eliminates the sink for the Main Feed Pump (see Figure 4.3), pressure becomes backed up from the

FWCV to the pump. Second, no pressure and no flow is propagated from the FWCV through the Manual Check Valve and into the boiler.

No flow into the boiler with normal flow out results in a low steam drum water level. Water level continues to drop until it reaches LOW_ALARM (indicating a Low Water casualty). Since there is virtually no more water in the steam drum or water drum at this point, the generation tubes will rupture (because they have no cooling medium). Steam drum pressure drops as a result of the tube rupture, and this reduced steam drum pressure (first LOW and then NONE), is propagated through all the steam piping systems. The periscope becomes fogged from the escaping steam in the furnace.

The volume of steam through the superheater and desuperheater drops to zero, and this change in flow is also propagated through the steam systems. Boiler fires are still lit and the superheater has no steam flowing through it. Like the generating tubes, it requires this flow to carry heat away. Without the flow, the superheater also ruptures.

The chain of events which occurred when the Feedwater Control Valve was shut accurately reflects what could happen in a real-world boiler if the same valve were shut and not immediate or controlling actions were taken.

VI. CONCLUSIONS

Model-based reasoning is an effective and efficient method for implementing steam plant engineering training. BoilerModel accurately represents physical components and executes concurrent real-world activities. It provides a usable output that shows how values are propagated through various systems. Answers to the questions posed in the thesis introduction will be examined in this chapter, along with other observations/ problems encountered, and recommendations for future work in this area.

A. QUESTIONS ANSWERED

1. Boiler and Steam Plant Modeling

Expert systems can be developed to efficiently and effectively model a propulsion boiler system. Rule-based systems could conceivably be built to correctly diagnose all casualty situations, but they have several shortcomings. First, they cannot reason beyond the limits of their rule bases. They are, therefore, limited by the knowledge of subject matter experts. Second, the number of rules required for such a system would be enormous because a great deal of them would be required to verify sensor accuracy. Moreover, the large number of rules would slow the expert system down (possibly to the point of uselessness). Third, modifications to the system (including expansion into a steam plant-wide domain) could require substantial alterations to the rule base. Such changes might result in redundant or contradictory rules.

BoilerModel is a streamlined expert system that does not rely on a bank of rules to determine plant status. Instead, it uses cause-effect relations and intra-component behavioral rules to propagate values to their logical conclusions. Since BoilerModel places

all emphasis on components and propagation along component connections, modification is simply a matter of modeling new devices and connecting them into the existing system. Therefore, expanding BoilerModel into a larger steam plant model is only as difficult as modeling the additional components.

2. Qualitative Modeling

Designing BoilerModel in the qualitative paradigm posed no problems. In fact, it may have been easier than doing so using actual FF 1052/1078 plant parameters because inexact state descriptions (HIGH, NORM, etc.) require no complicated mathematical formulae. Moreover, qualitative modeling of this project carries two advantages over mathematical or numerical modeling. First, BoilerModel can be used effectively as is by engineering personnel assigned to ships with different types of propulsion boilers and different plant configurations. The *general* sequence of events that occurs when the feedwater inlet to the boiler is shut is the same for all steam propulsion plants; only the parameters vary. Additionally, because of the component-oriented nature of model-based systems in general and the modularity of BoilerModel in particular, the code can be manipulated to add and remove components or to rearrange valves and piping configurations with very little difficulty. So, although BoilerModel is based on a frigate's steam generation system, it can be modified to match any other steam platform.

The second advantage a qualitative boiler model has over a mathematical one is that the real-world users of the model are plant operators, not mechanical engineers. Although both officer and enlisted watchstanders must know some plant-specific parameters, no one is required to know all of them. One reason is that there are *many* measurable parameters. Instead of requiring an operator to remember them all (and possibly forget some), engineering guidelines dictate the use of markers (such as red tape) on measuring devices (gauges and thermometers) to indicate the maximum acceptable high

or low values. A watchstander can then scan his or her gauge board and observe the relationship of the actual value to the max (or min) for that sensor. In other words, the watchstander makes *qualitative* observations; values are low, high, or normal with respect to the delimiting marker.

3. Modeling in Ada

Ada proved to be a versatile modeling tool. It provided fairly tight and very fast code. It can be used procedurally or functionally, and is very portable. Lisp code, on the other hand, ran considerably slower than Ada code in the case-based comparison. Moreover, the Lisp code proved more difficult to troubleshoot because it produced run-time errors which, while traceable (using the Lisp "trace" function), were not nearly as easy to locate and correct as compile-time errors in Ada.

Lisp is one of the dominant expert system modeling languages. Should it remain so? To answer *yes*, a Lisp proponent must provide clear advantages for that language over other contenders. This thesis proposes Ada as a language for use in the *full development* of artificial intelligence applications, from prototype to finished product. The only thing Ada lacks is true inheritance in object-oriented programming. That is only temporary; at least one preprocessor, Classic-Ada, allows full use of object-oriented techniques. When tools such as this one are widely available and become a *de facto* part of the DoD standard, then Ada will truly be an *all-purpose* language.

4. Affordability

Since BoilerModel was developed in Ada, affordability for shipboard use is based on three considerations. First, the initial purchase of an Ada compiler for the PC or Macintosh that can handle the large, numerous global variables inherent in the model. Since Ada is so portable, very little modification to existing code would be required for a changeover from UNIX to MS/PC-DOS or the Macintosh operating system. Second, ships

must be equipped with the hardware necessary to run the executable version of BoilerModel. Specifically, PC's or Mac's (preferably laptop versions) need to be accessible to engineering personnel. Third, if this model is to grow any larger than it is, someone needs to make it happen.

The first two considerations involve minimal costs that can easily be reconciled in any budget. The third consideration may involve man-hours (years) diverted toward project development, although some costs can be defrayed by using available research institutions (such as the Naval Postgraduate School).

B. OTHER OBSERVATIONS AND PROBLEMS

Observations were made and problems encountered during the design and development of BoilerModel that were not directly tied to the thesis questions. They will be discussed in this section.

1. Compiler Problems

As detailed in an earlier chapter, BoilerModel's complement of global variables prevented the use of Meridian's standard sized Ada compiler for the PC. This proved to be only a temporary snag; Ada's portability resulted in trouble free transfer to the Verdix Ada compiler on the Suns (although the user-friendly Meridian editing environment was sorely missed).

2. Incomplete Model

The boiler is probably the single most complicated component to model in the steam plant. There are several valves and subsystems that exist in real-world boiler systems but have not been built into BoilerModel. The reason for this lies with the goal to get a *working* model in Ada completed first. The supporting boiler systems (most notably the Automatic Boiler Control systems) can be added later. To its credit, BoilerModel provides

a detailed representation of a propulsion boiler that accurately propagates value changes to their logical conclusions.

3. Naval Reserve Training

Roughly twelve FF-1052 class frigates are scheduled for reclassification as "FT," or Frigate Trainers. Their function will be to train Naval Reservists on their weekend drills in a non-adversarial environment. Since the typical reservist is not exposed to more than roughly sixteen hours of shipboard duties per month, a portable, computerized trainer could maximize casualty control training while minimizing well-intentioned "mistakes" typical of undertrained deckplate sailors.

4. OPPE/LOE

All ships must undergo two periodic engineering inspections: the Operational Propulsion Plant Examination (OPPE) and the Light-Off Examination (LOE). Normally, sufficient underway time is allotted for OPPE preparation; however, since part of the exam is materiel readiness, a substantial amount of work at a repair facility is also required. Light-Off Exams are required after extensive yard periods, during which the ship cannot get underway to conduct realistic training. A shipboard training complement like BoilerModel could constructively use the inport time to prepare watchstanders for these inspections.

5. Other Propulsion Plants

Steam ships are fast becoming an anomaly. With the decommissioning of the Knox class frigates, the Adams and Farragut/Coontz class guided missile destroyers, and the battleships, the only steam driven platforms left will be auxiliaries, cruisers, some amphibious ships, and a handful of aircraft carriers. However, the concepts of model-based

design employed in BoilerModel transcend propulsion type and can therefore by applied to both gas turbine and diesel plants.

C. FUTURE WORK

Future work in BoilerModel can extend in several (non-mutually exclusive) directions. First, the boiler itself should be modeled in greater detail, incorporating subsystems (ABC, ACC, etc.) not included in this implementation. This would provide greater reliability as a diagnostic tool and greater accuracy as a training system.

A natural follow-on to the first suggestion would be to expand the scope of BoilerModel to cover the entire engineering plant. The groundwork for value propagation through valves and piping systems has already been laid, and the most complex component (the boiler) has already been modeled. Pumps can be modeled as classes in an object-oriented approach. This would be very useful because, although all pumps share certain characteristics, different types (centrifugal, positive displacement, jet) move fluids differently and therefore have varying intra-component relationships.

A third suggestion for future research is to incorporate BoilerModel into some sort of training or diagnostic system usable in the fleet (hopefully while there are still steam plants to operate). Since BoilerModel relies on model-based reasoning principles, little time need be spent interviewing experts regarding boiler operations. Energies can be devoted to developing an effective student model and a useful student interface.

A final suggestion for future work would be to incorporate BoilerModel (and any improvements to it) into the Argos paperless ship project as a stand-alone training module. The focus for this work could be a graphics and sound replacement for the textual user interface presently implemented. Boiler operation would then be realistically modeled *and* realistically presented. As a side effect, a sharp video presentation would probably enhance the appeal of the model to potential users.

APPENDIX A (BoilerModel CODE)

```
with SYSTEM;
use SYSTEM;
package VALVES_AND_PIPING is

type INDEX is (VSH, MSS, ASS, DESUP_IN, SH_PROT_IN,
                SAFETIES,AIR_REG, SHUTTERS, FUEL_MAN, FOCV,
                FO_RECIRC, FOSP_DISCH, FEED_STOP, FWCV,
                MAN_CHK, BLEED, AUG, FDB_IN, ONE_FIFTY_IN,
                PMAC_IN, ER_BLKHD_STOP, AMR_BLKHD_STOP,
                MFP_STM_SUP);

  type MEASUREMENT_VALUE is (NONE, LOW_ALARM, LOW, NORM,
                HIGH, LIFT_SAFE_HI, HIGH_ALARM, FUEL_ON_DECK,
                NO_FUEL_ON_DECK, ORANGE, BLACK, CLEAR,
                FOGGED,  FAN_SHAPED, IRREGULAR, OPEN, SHUT);
  type VALVE;
  type VALVE_PTR is access VALVE;
  type VALVE_PTR_ARRAY is array (1..10) of VALVE_PTR;

  type SYSTEM_ARRAY is array (POSITIVE range <>) of VALVE_PTR;

  TRACE: BOOLEAN:= FALSE;

  type SYSTEM_ARRAY_PTR is access SYSTEM_ARRAY;

  STEAM_SYSTEM: SYSTEM_ARRAY_PTR:= new SYSTEM_ARRAY(1..14);
 AIR_SYSTEM: SYSTEM_ARRAY_PTR:= new SYSTEM_ARRAY(1..2);
FUEL_SYSTEM: SYSTEM_ARRAY_PTR:= new SYSTEM_ARRAY(1..4);
FEED_SYSTEM: SYSTEM_ARRAY_PTR:= new SYSTEM_ARRAY(1..3);

  type VALVE is
    record
          VALVE_ID: INDEX;
          UPSTREAM: VALVE_PTR_ARRAY;
          DOWNSTREAM: VALVE_PTR_ARRAY;
          COUNTED: BOOLEAN:= FALSE;
          NEXT: VALVE_PTR;
          PREV_STATUS: MEASUREMENT_VALUE:= OPEN;
          STATUS: MEASUREMENT_VALUE:= OPEN;
          PREV_INPUT_PRESS: MEASUREMENT_VALUE:= NORM;
          INPUT_PRESSURE: MEASUREMENT_VALUE:= NORM;
          PREV_INPUT_FLOW: MEASUREMENT_VALUE:= NORM;
          PREV_OUTPUT_FLOW: MEASUREMENT_VALUE:= NORM;
          INPUT_FLOW: MEASUREMENT_VALUE:= NORM;
          OUTPUT_PRESSURE: MEASUREMENT_VALUE:= NORM;
          PREV_OUTPUT_PRESS: MEASUREMENT_VALUE:= NORM;
```

```
            OUTPUT_FLOW: MEASUREMENT_VALUE:= NORM;
            SYSTEM: SYSTEM_ARRAY_PTR;
        end record;

    VIRTUAL_SH_OUTLET: VALVE_PTR:= new VALVE;
    MAIN_STEAM_STOP: VALVE_PTR:= new VALVE;
    AUX_STEAM_STOP: VALVE_PTR:= new VALVE;
    DESUPERHEATER_INLET: VALVE_PTR:= new VALVE;
    SUPERHEATER_PROTECTION_INLET: VALVE_PTR:= new VALVE;
    SAFETY_VALVES: VALVE_PTR:= new VALVE;
    AIR_REGISTERS: VALVE_PTR:= new VALVE;
    AIR_SHUTTERS: VALVE_PTR:= new VALVE;
    FUEL_MANIFOLD: VALVE_PTR:= new VALVE;
    FUEL_OIL_CONTROL_VALVE: VALVE_PTR:= new VALVE;
    FUEL_RECIRC: VALVE_PTR:= new VALVE;
    FO_SVC_PUMP_DISCH: VALVE_PTR:= new VALVE;
    FEED_STOP_VALVE: VALVE_PTR:= new VALVE;
    FEEDWATER_CONTROL: VALVE_PTR:= new VALVE;
    MANUAL_CHECK_VLV: VALVE_PTR:= new VALVE;
    BLEEDER: VALVE_PTR:= new VALVE;
    AUGMENTOR: VALVE_PTR:= new VALVE;
    FDB_INLET: VALVE_PTR:= new VALVE;
    ONE_FIFTY_INLET: VALVE_PTR:= new VALVE;
    PMAC_INLET: VALVE_PTR:= new VALVE;
    ER_BULKHEAD_STOP: VALVE_PTR:= new VALVE;
    AMR_BULKHEAD_STOP: VALVE_PTR:= new VALVE;
    MFP_STM_SUPPLY: VALVE_PTR:= new VALVE;

    procedure FILL_SYSTEM_ARRAYS;
    procedure ASSIGN_VALVE_ORDER;
    procedure SHOW_VALVE_STATUS (AFFECTED: in out VALVE_PTR);
    procedure INITIALIZE_PLANT;
    procedure CHECK_FOR_FLOW (VALVE_IN: in out VALVE_PTR;
                              FLOW: in out BOOLEAN);
    procedure OVER_PRESSURE (AFFECTED: in out VALVE_PTR);
    task PROPAGATE is
        pragma PRIORITY(1);
    end PROPAGATE;

end VALVES_AND_PIPING;
```

```
with TEXT_IO;
use TEXT_IO;

package body VALVES_AND_PIPING is

package INDEX_IO is new ENUMERATION_IO (INDEX);
use INDEX_IO;

package MEASUREMENT_IO is new ENUMERATION_IO
        (MEASUREMENT_VALUE);
use MEASUREMENT_IO;


-------------------------------------------------------------------------
-- Procedure FILL_SYSTEM_ARRAYS fills steam, fuel, air
-- and feed system arrays with the valves in those systems
-------------------------------------------------------------------------

procedure FILL_SYSTEM_ARRAYS is

begin
  STEAM_SYSTEM(1):= VIRTUAL_SH_OUTLET;
  STEAM_SYSTEM(2):= MAIN_STEAM_STOP;
  STEAM_SYSTEM(3):= AUX_STEAM_STOP;
  STEAM_SYSTEM(4):= DESUPERHEATER_INLET;
  STEAM_SYSTEM(5):= SUPERHEATER_PROTECTION_INLET;
  STEAM_SYSTEM(6):= SAFETY_VALVES;
  STEAM_SYSTEM(7):= BLEEDER;
  STEAM_SYSTEM(8):= AUGMENTOR;
  STEAM_SYSTEM(9):= FDB_INLET;
  STEAM_SYSTEM(10):= ONE_FIFTY_INLET;
  STEAM_SYSTEM(11):= PMAC_INLET;
  STEAM_SYSTEM(12):= ER_BULKHEAD_STOP;
  STEAM_SYSTEM(13):= AMR_BULKHEAD_STOP;
  STEAM_SYSTEM(14):= MFP_STM_SUPPLY;

  AIR_SYSTEM(1):= AIR_SHUTTERS;
  AIR_SYSTEM(2):= AIR_REGISTERS;

  FUEL_SYSTEM(1):= FO_SVC_PUMP_DISCH;
  FUEL_SYSTEM(2):= FUEL_OIL_CONTROL_VALVE;
  FUEL_SYSTEM(3):= FUEL_RECIRC;
  FUEL_SYSTEM(4):= FUEL_MANIFOLD;

  FEED_SYSTEM(1):= FEED_STOP_VALVE;
  FEED_SYSTEM(2):= FEEDWATER_CONTROL;
  FEED_SYSTEM(3):= MANUAL_CHECK_VLV;
end FILL_SYSTEM_ARRAYS;
```

```
-------------------------------------------------------------------------------
-- Procedure ASSIGN_VALVE_ORDER fill the upstream and
-- downstream arrays of each valve with those valves which
-- are immediately upstream and downstream
-- (respectively)
-------------------------------------------------------------------------------

procedure ASSIGN_VALVE_ORDER is

begin

  VIRTUAL_SH_OUTLET.DOWNSTREAM(1):= MAIN_STEAM_STOP;
  VIRTUAL_SH_OUTLET.DOWNSTREAM(2):= DESUPERHEATER_INLET;
  MAIN_STEAM_STOP.UPSTREAM(1):= VIRTUAL_SH_OUTLET;
  MAIN_STEAM_STOP.DOWNSTREAM(1):= ER_BULKHEAD_STOP;
  MAIN_STEAM_STOP.DOWNSTREAM(2):= AMR_BULKHEAD_STOP;
  MAIN_STEAM_STOP.DOWNSTREAM(3):= MFP_STM_SUPPLY;
  DESUPERHEATER_INLET.UPSTREAM(1):= VIRTUAL_SH_OUTLET;
  DESUPERHEATER_INLET.DOWNSTREAM(1):= AUX_STEAM_STOP;
  DESUPERHEATER_INLET.DOWNSTREAM(2):= BLEEDER;
  AUX_STEAM_STOP.UPSTREAM(1):= DESUPERHEATER_INLET;
  AUX_STEAM_STOP.DOWNSTREAM(1):= AUGMENTOR;
  AUX_STEAM_STOP.DOWNSTREAM(2):= FDB_INLET;
  AUX_STEAM_STOP.DOWNSTREAM(3):= ONE_FIFTY_INLET;
  AUX_STEAM_STOP.DOWNSTREAM(5):= PMAC_INLET;
  AIR_REGISTERS.UPSTREAM(1):= AIR_SHUTTERS;
  AIR_SHUTTERS.DOWNSTREAM(1):= AIR_REGISTERS;
  FUEL_MANIFOLD.UPSTREAM(1):= FUEL_OIL_CONTROL_VALVE;
  FUEL_OIL_CONTROL_VALVE.UPSTREAM(1):= FO_SVC_PUMP_DISCH;
  FUEL_OIL_CONTROL_VALVE.DOWNSTREAM(1):= FUEL_MANIFOLD;
  FUEL_OIL_CONTROL_VALVE.DOWNSTREAM(2):= FUEL_RECIRC;
  FUEL_RECIRC.UPSTREAM(1):= FUEL_OIL_CONTROL_VALVE;
  FUEL_RECIRC.UPSTREAM(2):= FUEL_MANIFOLD;
  FO_SVC_PUMP_DISCH.DOWNSTREAM(1):= FUEL_OIL_CONTROL_VALVE;
  FEED_STOP_VALVE.DOWNSTREAM(1):= FEEDWATER_CONTROL;
  FEEDWATER_CONTROL.UPSTREAM(1):= FEED_STOP_VALVE;
  FEEDWATER_CONTROL.DOWNSTREAM(1):= MANUAL_CHECK_VLV;
  MANUAL_CHECK_VLV.UPSTREAM(1):= FEEDWATER_CONTROL;
  BLEEDER.UPSTREAM(1):= DESUPERHEATER_INLET;
  AUGMENTOR.UPSTREAM(1):= AUX_STEAM_STOP;
  FDB_INLET.UPSTREAM(1):= AUX_STEAM_STOP;
  ONE_FIFTY_INLET.UPSTREAM(1):= AUX_STEAM_STOP;
  PMAC_INLET.UPSTREAM(1):= AUX_STEAM_STOP;
  ER_BULKHEAD_STOP.UPSTREAM(1):= MAIN_STEAM_STOP;
  AMR_BULKHEAD_STOP.UPSTREAM(1):= MAIN_STEAM_STOP;
  MFP_STM_SUPPLY.UPSTREAM(1):= MAIN_STEAM_STOP;

end ASSIGN_VALVE_ORDER;
```

```
-------------------------------------------------------------------------------
-- procedure SHOW_VALVE_STATUS provides a trace to the user
-- (if desired) to show the effects of propagation after a system change
-- is made --
-------------------------------------------------------------------------------

procedure SHOW_VALVE_STATUS (AFFECTED: in out VALVE_PTR) is

  ROW_COUNT: NATURAL:= 1;

begin
  NEW_LINE;
  PUT("VALVE");
  SET_COL(17);
  PUT("STATUS");
  SET_COL(24);
  PUT("INPUT PRESS");
  SET_COL(40);
  PUT("INPUT FLOW");
  SET_COL(52);
  PUT("OUTPUT PRESS");
  SET_COL(67);
  PUT_LINE("OUTPUT FLOW");
  PUT("-----");
  SET_COL(17);
  PUT("------");
  SET_COL(24);
  PUT("----------");
  SET_COL(40);
  PUT("----------");
  SET_COL(52);
  PUT("-----------");
  SET_COL(67);
  PUT("----------");
  NEW_LINE;
  PUT(AFFECTED.VALVE_ID);
  SET_COL(17);
  PUT(AFFECTED.STATUS);
  SET_COL(24);
  PUT(AFFECTED.INPUT_PRESSURE);
  SET_COL(40);
  PUT(AFFECTED.INPUT_FLOW);
  SET_COL(52);
  PUT(AFFECTED.OUTPUT_PRESSURE);
  SET_COL(67);
  PUT(AFFECTED.OUTPUT_FLOW);
  NEW_LINE;
  delay 0.0;
end SHOW_VALVE_STATUS;
```

```
-------------------------------------------------------------------------------
-- Procedure INITIALIZE_PLANT sets the valves in the plant to
-- the correct setting for steady-state steaming.  It also identifies each
-- valve with an ID (for printing or reference) and identifies the
-- system with which the valve is associated
-------------------------------------------------------------------------------

procedure INITIALIZE_PLANT is

begin
  VIRTUAL_SH_OUTLET.VALVE_ID:= VSH;
  VIRTUAL_SH_OUTLET.NEXT:= MAIN_STEAM_STOP;
  VIRTUAL_SH_OUTLET.SYSTEM:= STEAM_SYSTEM;
  MAIN_STEAM_STOP.VALVE_ID:= MSS;
  MAIN_STEAM_STOP.NEXT:= AUX_STEAM_STOP;
  MAIN_STEAM_STOP.SYSTEM:= STEAM_SYSTEM;
  AUX_STEAM_STOP.VALVE_ID:= ASS;
  AUX_STEAM_STOP.NEXT:= DESUPERHEATER_INLET;
  AUX_STEAM_STOP.SYSTEM:= STEAM_SYSTEM;
  DESUPERHEATER_INLET.VALVE_ID:= DESUP_IN;
  DESUPERHEATER_INLET.NEXT:= SUPERHEATER_PROTECTION_INLET;
  DESUPERHEATER_INLET.SYSTEM:= STEAM_SYSTEM;
  SUPERHEATER_PROTECTION_INLET.VALVE_ID:= SH_PROT_IN;
  SUPERHEATER_PROTECTION_INLET.NEXT:= SAFETY_VALVES;
  SUPERHEATER_PROTECTION_INLET.PREV_STATUS:= SHUT;
  SUPERHEATER_PROTECTION_INLET.STATUS:= SHUT;
  SUPERHEATER_PROTECTION_INLET.SYSTEM:= STEAM_SYSTEM;
  SUPERHEATER_PROTECTION_INLET.OUTPUT_FLOW:= NONE;
  SUPERHEATER_PROTECTION_INLET.PREV_OUTPUT_FLOW:= NONE;
  SUPERHEATER_PROTECTION_INLET.OUTPUT_PRESSURE:= NONE;
  SUPERHEATER_PROTECTION_INLET.PREV_OUTPUT_PRESS:= NONE;
  SAFETY_VALVES.VALVE_ID:= SAFETIES;
  SAFETY_VALVES.NEXT:= AIR_REGISTERS;
  SAFETY_VALVES.PREV_STATUS:= SHUT;
  SAFETY_VALVES.STATUS:= SHUT;
  SAFETY_VALVES.SYSTEM:= STEAM_SYSTEM;
  SAFETY_VALVES.OUTPUT_FLOW:= NONE;
  SAFETY_VALVES.PREV_OUTPUT_FLOW:= NONE;
  SAFETY_VALVES.OUTPUT_PRESSURE:= NONE;
  SAFETY_VALVES.PREV_OUTPUT_PRESS:= NONE;
  AIR_REGISTERS.VALVE_ID:= AIR_REG;
  AIR_REGISTERS.NEXT:= AIR_SHUTTERS;
  AIR_REGISTERS.SYSTEM:= AIR_SYSTEM;
  AIR_SHUTTERS.VALVE_ID:= SHUTTERS;
  AIR_SHUTTERS.NEXT:= FUEL_MANIFOLD;
  AIR_SHUTTERS.SYSTEM:= AIR_SYSTEM;
  FUEL_MANIFOLD.VALVE_ID:= FUEL_MAN;
  FUEL_MANIFOLD.NEXT:= FUEL_OIL_CONTROL_VALVE;
  FUEL_MANIFOLD.SYSTEM:= FUEL_SYSTEM;
  FUEL_OIL_CONTROL_VALVE.VALVE_ID:= FOCV;
  FUEL_OIL_CONTROL_VALVE.NEXT:= FUEL_RECIRC;
  FUEL_OIL_CONTROL_VALVE.SYSTEM:= FUEL_SYSTEM;
```

```
FUEL_RECIRC.VALVE_ID:= FO_RECIRC;
FUEL_RECIRC.NEXT:= FO_SVC_PUMP_DISCH;
FUEL_RECIRC.PREV_STATUS:= SHUT;
FUEL_RECIRC.STATUS:= SHUT;
FUEL_RECIRC.OUTPUT_FLOW:= NONE;
FUEL_RECIRC.PREV_OUTPUT_FLOW:= NONE;
FUEL_RECIRC.OUTPUT_PRESSURE:= NONE;
FUEL_RECIRC.PREV_OUTPUT_PRESS:= NONE;
FUEL_RECIRC.SYSTEM:= FUEL_SYSTEM;
FO_SVC_PUMP_DISCH.VALVE_ID:= FOSP_DISCH;
FO_SVC_PUMP_DISCH.NEXT:= FEED_STOP_VALVE;
FO_SVC_PUMP_DISCH.SYSTEM:= FUEL_SYSTEM;
FEED_STOP_VALVE.VALVE_ID:= FEED_STOP;
FEED_STOP_VALVE.NEXT:= FEEDWATER_CONTROL;
FEED_STOP_VALVE.SYSTEM:= FEED_SYSTEM;
FEEDWATER_CONTROL.VALVE_ID:= FWCV;
FEEDWATER_CONTROL.NEXT:= MANUAL_CHECK_VLV;
FEEDWATER_CONTROL.SYSTEM:= FEED_SYSTEM;
MANUAL_CHECK_VLV.VALVE_ID:= MAN_CHK;
MANUAL_CHECK_VLV.NEXT:= BLEEDER;
MANUAL_CHECK_VLV.SYSTEM:= FEED_SYSTEM;
BLEEDER.VALVE_ID:= BLEED;
BLEEDER.NEXT:= AUGMENTOR;
BLEEDER.PREV_STATUS:= SHUT;
BLEEDER.STATUS:= SHUT;
BLEEDER.OUTPUT_FLOW:= NONE;
BLEEDER.PREV_OUTPUT_FLOW:= NONE;
BLEEDER.OUTPUT_PRESSURE:= NONE;
BLEEDER.PREV_OUTPUT_PRESS:= NONE;
BLEEDER.SYSTEM:= STEAM_SYSTEM;
AUGMENTOR.VALVE_ID:= AUG;
AUGMENTOR.NEXT:= FDB_INLET;
AUGMENTOR.SYSTEM:= STEAM_SYSTEM;
FDB_INLET.VALVE_ID:= FDB_IN;
FDB_INLET.NEXT:= ONE_FIFTY_INLET;
FDB_INLET.SYSTEM:= STEAM_SYSTEM;
ONE_FIFTY_INLET.VALVE_ID:= ONE_FIFTY_IN;
ONE_FIFTY_INLET.NEXT:= PMAC_INLET;
ONE_FIFTY_INLET.SYSTEM:= STEAM_SYSTEM;
PMAC_INLET.VALVE_ID:= PMAC_IN;
PMAC_INLET.NEXT:= ER_BULKHEAD_STOP;
PMAC_INLET.PREV_STATUS:= SHUT;
PMAC_INLET.STATUS:= SHUT;
PMAC_INLET.OUTPUT_FLOW:= NONE;
PMAC_INLET.PREV_OUTPUT_FLOW:= NONE;
PMAC_INLET.OUTPUT_PRESSURE:= NONE;
PMAC_INLET.PREV_OUTPUT_PRESS:= NONE;
PMAC_INLET.SYSTEM:= STEAM_SYSTEM;
ER_BULKHEAD_STOP.VALVE_ID:= ER_BLKHD_STOP;
ER_BULKHEAD_STOP.NEXT:= AMR_BULKHEAD_STOP;
ER_BULKHEAD_STOP.SYSTEM:= STEAM_SYSTEM;
AMR_BULKHEAD_STOP.VALVE_ID:= AMR_BLKHD_STOP;
```

```
  AMR_BULKHEAD_STOP.NEXT:= MFP_STM_SUPPLY;
  AMR_BULKHEAD_STOP.SYSTEM:= STEAM_SYSTEM;
  MFP_STM_SUPPLY.VALVE_ID:= MFP_STM_SUP;
  MFP_STM_SUPPLY.NEXT:= VIRTUAL_SH_OUTLET;
  MFP_STM_SUPPLY.SYSTEM:= STEAM_SYSTEM;
end INITIALIZE_PLANT;

-------------------------------------------------------------------------------
-- Procedure CHECK_FOR_FLOW ascertains whether or not
-- there is still a flow path in a system after a valve's status has
-- been changed
-------------------------------------------------------------------------------

procedure CHECK_FOR_FLOW(VALVE_IN: in out VALVE_PTR;
   FLOW: in out BOOLEAN) is

 COUNT, COUNT2: NATURAL:= 1;

begin
 VALVE_IN.COUNTED:= TRUE;
 if VALVE_IN.STATUS = OPEN and VALVE_IN.DOWNSTREAM(1) = null then
  FLOW:= TRUE;
 else
  if VALVE_IN.STATUS = OPEN then
    while VALVE_IN.DOWNSTREAM(COUNT) /= null loop
      if VALVE_IN.DOWNSTREAM(COUNT).COUNTED = FALSE then
        CHECK_FOR_FLOW(VALVE_IN.DOWNSTREAM(COUNT),FLOW);
      end if;
      COUNT:= COUNT + 1;
    end loop;
  end if;
  if FLOW = FALSE then
    while VALVE_IN.UPSTREAM(COUNT2) /= null loop
      CHECK_FOR_FLOW(VALVE_IN.UPSTREAM(COUNT2), FLOW);
      COUNT2:= COUNT2 + 1;
   end loop;
  end if;
 end if;
end CHECK_FOR_FLOW;

-------------------------------------------------------------------------------
-- Procedure OVER_PRESSURE will create an overpressurization
-- in a system which is supplied pressure but which has no flow
-------------------------------------------------------------------------------

procedure OVER_PRESSURE(AFFECTED: in out VALVE_PTR) is

 COUNT: NATURAL:= 1;

begin
 if AFFECTED.INPUT_PRESSURE < HIGH and AFFECTED.VALVE_ID /= SAFETIES then
   AFFECTED.INPUT_PRESSURE:= HIGH;
```

```
    while AFFECTED.UPSTREAM(COUNT) /= null loop
      AFFECTED.UPSTREAM(COUNT).OUTPUT_PRESSURE:= HIGH;
      if AFFECTED.UPSTREAM(COUNT).STATUS = OPEN then
        OVER_PRESSURE(AFFECTED.UPSTREAM(COUNT));
      end if;
      COUNT:= COUNT + 1;
    end loop;
     if TRACE = TRUE then
        SHOW_VALVE_STATUS(AFFECTED);
    end if;
    AFFECTED.PREV_INPUT_PRESS:= AFFECTED.INPUT_PRESSURE;
  end if;
end OVER_PRESSURE;

-------------------------------------------------------------------------------
-- Procedure PROPAGATE_VALVE_VALUES propagates
-- new system values through the  upstream/downstream lattice
-- when a valve's status has been changed.  It calls
-- CHECK_FOR_FLOW and OVER_PRESSURE (as needed)
-------------------------------------------------------------------------------


procedure PROPAGATE_VALVE_VALUES (AFFECTED: in out VALVE_PTR) is

  COUNT: NATURAL:= 1;
  FLOW: BOOLEAN:= FALSE;

begin
  if AFFECTED.STATUS = OPEN then
     AFFECTED.OUTPUT_PRESSURE:= AFFECTED.INPUT_PRESSURE;
     AFFECTED.OUTPUT_FLOW:= AFFECTED.INPUT_FLOW;
     if TRACE = TRUE then
        SHOW_VALVE_STATUS(AFFECTED);
     end if;
  else
     AFFECTED.OUTPUT_FLOW:= NONE;
     AFFECTED.OUTPUT_PRESSURE:= NONE;
     if TRACE = TRUE then
        SHOW_VALVE_STATUS(AFFECTED);
     end if;
     CHECK_FOR_FLOW(AFFECTED, FLOW);
     if FLOW = FALSE then
        OVER_PRESSURE(AFFECTED);
     end if;
     for i in 1..AFFECTED.SYSTEM'LENGTH loop
        AFFECTED.SYSTEM(i).COUNTED:= FALSE;
     end loop;
  end if;
  AFFECTED.PREV_STATUS:= AFFECTED.STATUS;
  AFFECTED.PREV_INPUT_PRESS:= AFFECTED.INPUT_PRESSURE;
  AFFECTED.PREV_INPUT_FLOW:= AFFECTED.INPUT_FLOW;
  while AFFECTED.DOWNSTREAM(COUNT) /= null loop
```

```
      AFFECTED.DOWNSTREAM(COUNT).INPUT_FLOW:= AFFECTED.OUTPUT_FLOW;
     AFFECTED.DOWNSTREAM(COUNT).INPUT_PRESSURE:=
          AFFECTED.OUTPUT_PRESSURE;
    COUNT:= COUNT + 1;
   end loop;

end PROPAGATE_VALVE_VALUES;

-------------------------------------------------------------------------------
-- Task PROPAGATE continuously monitors all valves in the
-- system and calls PROPAGATE_VALVE_VALUES whenever
--  input pressure, input flow, or output flow changes, or when
--  a valve's status is changed
-------------------------------------------------------------------------------

task body PROPAGATE is
    CURRENT: VALVE_PTR:= VIRTUAL_SH_OUTLET;
begin
  delay 1.0;
  loop
    if CURRENT.STATUS /= CURRENT.PREV_STATUS or
      CURRENT.INPUT_PRESSURE /= CURRENT.PREV_INPUT_PRESS or
      CURRENT.INPUT_FLOW /= CURRENT.PREV_INPUT_FLOW then
        PROPAGATE_VALVE_VALUES(CURRENT);
    end if;
    CURRENT:= CURRENT.NEXT;
    delay 0.01;
  end loop;
end PROPAGATE;

end VALVES_AND_PIPING;
```

```
with VALVES_AND_PIPING, SYSTEM;
use VALVES_AND_PIPING, SYSTEM;
package BOILER_MODEL is

type DRUM is
 record
   WATER_LEVEL: MEASUREMENT_VALUE;
   PRESSURE: MEASUREMENT_VALUE;
   FLOW_IN: MEASUREMENT_VALUE;
   FLOW_OUT: MEASUREMENT_VALUE;
   TEMP: MEASUREMENT_VALUE;
   PREV_WATER_LEVEL: MEASUREMENT_VALUE;
   PREV_PRESSURE: MEASUREMENT_VALUE;
   PREV_FLOW_IN: MEASUREMENT_VALUE;
   PREV_FLOW_OUT: MEASUREMENT_VALUE;
   PREV_TEMP: MEASUREMENT_VALUE;
 end record;

type TUBE is
 record
   FLOW_IN: MEASUREMENT_VALUE;
   FLOW_OUT: MEASUREMENT_VALUE;
   RUPTURE: BOOLEAN:= FALSE;
   TEMP: MEASUREMENT_VALUE;
   PREV_FLOW_IN: MEASUREMENT_VALUE;
   PREV_FLOW_OUT: MEASUREMENT_VALUE;
   PREV_RUPTURE: BOOLEAN:= FALSE;
   PREV_TEMP: MEASUREMENT_VALUE;
   DIRECT_HEAT_CONTACT: BOOLEAN;
 end record;

type BOILER_FURNACE is
 record
   DECK_STATUS: MEASUREMENT_VALUE;
   FIRING_RATE: MEASUREMENT_VALUE;
   EXPLOSION: BOOLEAN:= FALSE;
   PERISCOPE: MEASUREMENT_VALUE;
   FIRES_LIT: BOOLEAN:= FALSE;
   FIRE_APPEARANCE: MEASUREMENT_VALUE;
   PREV_DECK_STATUS: MEASUREMENT_VALUE;
   PREV_FIRING_RATE: MEASUREMENT_VALUE;
   PREV_EXPLOSION: BOOLEAN:= FALSE;
   PREV_PERISCOPE: MEASUREMENT_VALUE;
   PREV_FIRES_LIT: BOOLEAN:= FALSE;
   PREV_FIRE_APPEARANCE: MEASUREMENT_VALUE;
 end record;
```

```
type BOILER is
 record
   STEAM_DRUM: DRUM;
   WATER_DRUM: DRUM;
   SUPERHEATER: TUBE;
   DESUPERHEATER: TUBE;
   GENERATION_TUBES: TUBE;
   FURNACE: BOILER_FURNACE;
 end record;

ALPHA_BOILER: BOILER;

procedure INITIALIZE_BOILER (BLR: in out BOILER);
procedure INCREMENTAL_DECREASE (BLR: in out BOILER);
task FIRES_MANAGER is
   pragma PRIORITY(1);
end FIRES_MANAGER;
task STEAM_DRUM_MANAGER is
  pragma PRIORITY(1);
end STEAM_DRUM_MANAGER;
task TUBE_MANAGER is
  pragma PRIORITY(1);
end TUBE_MANAGER;
function GREATER_OF (X, Y: in MEASUREMENT_VALUE) return MEASUREMENT_VALUE;
function LESSER_OF (X, Y: in MEASUREMENT_VALUE) return MEASUREMENT_VALUE;
procedure RUPTURED_TUBE_CHECK (BLR: in out BOILER; X: in out TUBE);
procedure DISPLAY_BOILER_STATUS (BLR: in out BOILER);
procedure RESET_BOILER_PREVIOUS_VALUES (BLR: in out BOILER);

end BOILER_MODEL;
```

```
with TEXT_IO, VALVES_AND_PIPING, SYSTEM;
use TEXT_IO, VALVES_AND_PIPING, SYSTEM;
package body BOILER_MODEL is

package TRUTH is new ENUMERATION_IO(BOOLEAN);
use TRUTH;
package MEASURE is new ENUMERATION_IO(MEASUREMENT_VALUE);
use MEASURE;


-------------------------------------------------------------------------------
-- Task STEAM_DRUM_MANAGER takes the water flow into the
-- boiler, compares it  with steam flow out of the boiler and generates
-- high/low water level alarm  conditions based on that comparison.  It also
-- lifts safeties if there is no flow out of the boiler.  STEAM_DRUM_MANAGER
-- also calls procedure INCREMENTAL_DECREASE if safeties are open.
-------------------------------------------------------------------------------

task body STEAM_DRUM_MANAGER is

begin
  delay 1.0;
  loop
    ALPHA_BOILER.STEAM_DRUM.FLOW_IN:= MANUAL_CHECK_VLV.OUTPUT_FLOW;
    if ALPHA_BOILER.STEAM_DRUM.FLOW_IN = NORM and
          ALPHA_BOILER.STEAM_DRUM.FLOW_OUT = NORM then
      ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL:= NORM;
    elsif ALPHA_BOILER.STEAM_DRUM.FLOW_IN >
          VIRTUAL_SH_OUTLET.OUTPUT_FLOW then
      ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL:= HIGH;
      delay 0.5;
      if ALPHA_BOILER.STEAM_DRUM.FLOW_OUT = NONE then
        ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL:= HIGH_ALARM;
      end if;
    elsif ALPHA_BOILER.STEAM_DRUM.FLOW_IN <
          ALPHA_BOILER.STEAM_DRUM.FLOW_OUT
          or (ALPHA_BOILER.STEAM_DRUM.FLOW_IN = NONE and
          ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL > LOW) then
      ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL:= LOW;
      delay 0.5;
      if ALPHA_BOILER.STEAM_DRUM.FLOW_IN = NONE then
        ALPHA_BOILER.STEAM_DRUM.WATER_LEVEL:= LOW_ALARM;
      end if;
    end if;
    delay 2.0;
    if VIRTUAL_SH_OUTLET.INPUT_PRESSURE = HIGH then
      ALPHA_BOILER.STEAM_DRUM.PRESSURE:= LIFT_SAFE_HI;
      delay 0.0;
      SAFETY_VALVES.STATUS:= OPEN;
    end if;
    delay 2.0;
```

```
      SAFETY_VALVES.INPUT_PRESSURE:= ALPHA_BOILER.STEAM_DRUM.PRESSURE;
    VIRTUAL_SH_OUTLET.INPUT_PRESSURE:=
              ALPHA_BOILER.STEAM_DRUM.PRESSURE;
    delay 2.0;
    if SAFETY_VALVES.STATUS = OPEN then
      SAFETY_VALVES.INPUT_FLOW:= NORM;
      INCREMENTAL_DECREASE(ALPHA_BOILER);
    end if;
    delay 2.0;
  end loop;
end STEAM_DRUM_MANAGER;

-------------------------------------------------------------------------
-- Task FIRES_MANAGER controls firing rate by controlling the output
-- of the fuel oil control valve based on steam drum pressure
-------------------------------------------------------------------------

task body FIRES_MANAGER is

begin
  delay 1.0;
  loop
    if ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE then
      ALPHA_BOILER.FURNACE.FIRING_RATE:=
                  FUEL_MANIFOLD.OUTPUT_PRESSURE;
    end if;
    if ALPHA_BOILER.FURNACE.FIRING_RATE = NONE then
      ALPHA_BOILER.FURNACE.FIRES_LIT:= FALSE;
      ALPHA_BOILER.FURNACE.FIRE_APPEARANCE:= NONE;
      ALPHA_BOILER.STEAM_DRUM.TEMP:= LOW;
      ALPHA_BOILER.GENERATION_TUBES.TEMP:= LOW;
      ALPHA_BOILER.SUPERHEATER.TEMP:= LOW;
      ALPHA_BOILER.DESUPERHEATER.TEMP:= LOW;
    end if;
    if ALPHA_BOILER.FURNACE.FIRING_RATE = NONE then
      ALPHA_BOILER.STEAM_DRUM.PRESSURE:= LOW;
    elsif ALPHA_BOILER.GENERATION_TUBES.RUPTURE or
        ALPHA_BOILER.SUPERHEATER.RUPTURE or
        VIRTUAL_SH_OUTLET.INPUT_FLOW = NONE then
      if ALPHA_BOILER.STEAM_DRUM.PRESSURE > LOW then
        ALPHA_BOILER.STEAM_DRUM.PRESSURE:= LOW;
        delay 20.0;
      end if;
      ALPHA_BOILER.STEAM_DRUM.PRESSURE:= NONE;
    end if;
    if FUEL_MANIFOLD.STATUS = OPEN and
        FUEL_MANIFOLD.INPUT_PRESSURE /= NONE and
        ALPHA_BOILER.STEAM_DRUM.FLOW_IN /= NONE then
      if ALPHA_BOILER.STEAM_DRUM.PRESSURE > NORM then
        FUEL_OIL_CONTROL_VALVE.OUTPUT_PRESSURE:= LOW;
        delay 2.0;
        if ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE then
```

```
        if VIRTUAL_SH_OUTLET.INPUT_PRESSURE < HIGH then
          ALPHA_BOILER.STEAM_DRUM.PRESSURE:= NORM;
        end if;
      else
        ALPHA_BOILER.FURNACE.DECK_STATUS:= FUEL_ON_DECK;
        ALPHA_BOILER.FURNACE.PERISCOPE:= FOGGED;
      end if;
    elsif ALPHA_BOILER.STEAM_DRUM.PRESSURE < NORM then
      FUEL_OIL_CONTROL_VALVE.OUTPUT_PRESSURE:= HIGH;
      delay 2.0;
      if SAFETY_VALVES.STATUS /= OPEN and
            ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE then
        ALPHA_BOILER.STEAM_DRUM.PRESSURE:= NORM;
      end if;
      if ALPHA_BOILER.FURNACE.FIRES_LIT = FALSE then
        ALPHA_BOILER.FURNACE.DECK_STATUS:= FUEL_ON_DECK;
      end if;
    elsif ALPHA_BOILER.STEAM_DRUM.PRESSURE = NORM and
            FUEL_MANIFOLD.OUTPUT_PRESSURE /= NORM and
            FUEL_MANIFOLD.OUTPUT_PRESSURE /= NONE and
            FUEL_MANIFOLD.STATUS = OPEN then
      FUEL_OIL_CONTROL_VALVE.OUTPUT_PRESSURE:=
            LESSER_OF(LOW, FO_SVC_PUMP_DISCH.OUTPUT_PRESSURE);
    end if;
  end if;
  delay 0.0;
  if ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE and
          AIR_REGISTERS.OUTPUT_FLOW = NONE then
    ALPHA_BOILER.FURNACE.FIRE_APPEARANCE:= IRREGULAR;
    ALPHA_BOILER.FURNACE.PERISCOPE:= BLACK;
  end if;
  delay 0.0;
  if ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE and
          AIR_REGISTERS.OUTPUT_FLOW /= NONE and
          ALPHA_BOILER.FURNACE.DECK_STATUS = FUEL_ON_DECK then
    ALPHA_BOILER.FURNACE.EXPLOSION:= TRUE;
  end if;
  delay 0.0;
  if FUEL_MANIFOLD.OUTPUT_FLOW < AIR_REGISTERS.OUTPUT_FLOW then
    ALPHA_BOILER.FURNACE.PERISCOPE:= ORANGE;
    ALPHA_BOILER.FURNACE.FIRE_APPEARANCE:= IRREGULAR;
    ALPHA_BOILER.SUPERHEATER.TEMP:= LOW;
    delay 60.0;
    ALPHA_BOILER.FURNACE.EXPLOSION:= TRUE;
  end if;
  delay 0.0;
  if FUEL_MANIFOLD.OUTPUT_PRESSURE /= NONE and
          FUEL_MANIFOLD.OUTPUT_PRESSURE <
          AIR_REGISTERS.OUTPUT_PRESSURE and
          ALPHA_BOILER.FURNACE.FIRING_RATE /= NONE then
    ALPHA_BOILER.FURNACE.FIRE_APPEARANCE:= IRREGULAR;
```

```
      delay 10.0;
      ALPHA_BOILER.FURNACE.FIRING_RATE:= NONE;
    end if;
    delay 0.0;
  end loop;
end FIRES_MANAGER;

-------------------------------------------------------------------------
-- Procedure INCREMENTAL_DECREASE reduces steam drum pressure
-- in the event of safeties lifting
-------------------------------------------------------------------------

procedure INCREMENTAL_DECREASE (BLR: in out BOILER) is

begin
  loop
    case BLR.STEAM_DRUM.PRESSURE is
      when LIFT_SAFE_HI =>
        delay 2.0;
        BLR.STEAM_DRUM.PRESSURE:= HIGH;
        SAFETY_VALVES.INPUT_PRESSURE:= BLR.STEAM_DRUM.PRESSURE;
        delay 0.0;
      when HIGH =>
        delay 2.0;
        BLR.STEAM_DRUM.PRESSURE:= NORM;
        SAFETY_VALVES.INPUT_PRESSURE:= BLR.STEAM_DRUM.PRESSURE;
        SAFETY_VALVES.STATUS:= SHUT;
        SAFETY_VALVES.OUTPUT_FLOW:= NONE;
        delay 0.0;
        exit;
      when others =>
        null;
    end case;
  end loop;
end INCREMENTAL_DECREASE;

-------------------------------------------------------------------------
-- Function GREATER_OF compares two MEASUREMENT_VALUE's'
-- and returns the one with the greater value
-------------------------------------------------------------------------

function GREATER_OF (X, Y: in MEASUREMENT_VALUE) return
                     MEASUREMENT_VALUE is
  VALUE: MEASUREMENT_VALUE;

begin
  if X > Y then
    VALUE:= X;
  else
    VALUE:= Y;
  end if;
  return VALUE;
end GREATER_OF;
```

```
-------------------------------------------------------------------------
-- Function LESSER_OF compares two MEASUREMENT_VALUE's
-- and returns the one with the greater value
-------------------------------------------------------------------------

function LESSER_OF (X, Y: in MEASUREMENT_VALUE) return
                        MEASUREMENT_VALUE is
  VALUE: MEASUREMENT_VALUE;

begin
  if X < Y then
    VALUE:= X;
  else
    VALUE:= Y;
  end if;
  return VALUE;
end LESSER_OF;


-------------------------------------------------------------------------
-- Procedure RUPTURED_TUBE_CHECK looks for less input flow
-- to a tube than output flow, or when there is no input flow.  Since
-- flow through a tube cools the inner surfaces of the tube, no flow or
-- severely restricted flow will result in a ruptured tube
-------------------------------------------------------------------------

procedure RUPTURED_TUBE_CHECK (BLR: in out BOILER; X: in out TUBE) is

begin
  if BLR.FURNACE.FIRES_LIT then
    if X.FLOW_IN = NONE then
      delay 10.0;
      if X.FLOW_OUT = NONE then
        X.TEMP:= HIGH;
        if X.DIRECT_HEAT_CONTACT then
          X.RUPTURE:= TRUE;
        end if;
      end if;
    elsif X.FLOW_IN < X.FLOW_OUT then
      delay 20.0;
      if X.FLOW_IN < X.FLOW_OUT then
        X.TEMP:= HIGH;
        if X.DIRECT_HEAT_CONTACT then
          X.RUPTURE:= TRUE;
        end if;
      end if;
    end if;
    if X.RUPTURE then
      BLR.FURNACE.PERISCOPE:= FOGGED;
    end if;
  end if;
end RUPTURED_TUBE_CHECK;
```

```
-------------------------------------------------------------------------
-- Task TUBE_MANAGER controls flow out of the boiler, depending
-- on water input to tubes and whether or not a tube rupture has occurred
-------------------------------------------------------------------------

task body TUBE_MANAGER is

begin
  delay 1.0;
  loop
    ALPHA_BOILER.STEAM_DRUM.FLOW_IN:=
                  MANUAL_CHECK_VLV.OUTPUT_FLOW;
    ALPHA_BOILER.WATER_DRUM.FLOW_IN:=
                  ALPHA_BOILER.STEAM_DRUM.FLOW_IN;
    ALPHA_BOILER.WATER_DRUM.FLOW_OUT:=
                ALPHA_BOILER.WATER_DRUM.FLOW_IN;
    ALPHA_BOILER.GENERATION_TUBES.FLOW_IN:=
                ALPHA_BOILER.WATER_DRUM.FLOW_OUT;
    if ALPHA_BOILER.FURNACE.FIRING_RATE = NONE then
      delay 10.0;
      ALPHA_BOILER.GENERATION_TUBES.FLOW_OUT:= LOW;
    else
      ALPHA_BOILER.GENERATION_TUBES.FLOW_OUT:=
                ALPHA_BOILER.GENERATION_TUBES.FLOW_IN;
    end if;
    ALPHA_BOILER.STEAM_DRUM.FLOW_OUT:=
                ALPHA_BOILER.GENERATION_TUBES.FLOW_OUT;
    delay 1.0;
    RUPTURED_TUBE_CHECK (ALPHA_BOILER,
                ALPHA_BOILER.GENERATION_TUBES);
    ALPHA_BOILER.SUPERHEATER.FLOW_IN:=
                ALPHA_BOILER.STEAM_DRUM.FLOW_OUT;
    SAFETY_VALVES.INPUT_FLOW:= ALPHA_BOILER.STEAM_DRUM.FLOW_OUT;
    ALPHA_BOILER.SUPERHEATER.FLOW_OUT:=
                ALPHA_BOILER.SUPERHEATER.FLOW_IN;
    delay 1.0;
    RUPTURED_TUBE_CHECK (ALPHA_BOILER, ALPHA_BOILER.SUPERHEATER);
    VIRTUAL_SH_OUTLET.INPUT_FLOW:=
                ALPHA_BOILER.SUPERHEATER.FLOW_OUT;
    if VIRTUAL_SH_OUTLET.STATUS = OPEN then
      VIRTUAL_SH_OUTLET.OUTPUT_FLOW:=
                VIRTUAL_SH_OUTLET.INPUT_FLOW;
    else
      VIRTUAL_SH_OUTLET.OUTPUT_FLOW:= NONE;
    end if;
    ALPHA_BOILER.DESUPERHEATER.FLOW_IN:=
                VIRTUAL_SH_OUTLET.OUTPUT_FLOW;
    MAIN_STEAM_STOP.INPUT_FLOW:= VIRTUAL_SH_OUTLET.OUTPUT_FLOW;
    ALPHA_BOILER.DESUPERHEATER.FLOW_OUT:=
                ALPHA_BOILER.DESUPERHEATER.FLOW_IN;
    AUX_STEAM_STOP.INPUT_FLOW:=
                ALPHA_BOILER.DESUPERHEATER.FLOW_OUT;
```

```
    delay 1.0;
    RUPTURED_TUBE_CHECK (ALPHA_BOILER,
                         ALPHA_BOILER.DESUPERHEATER);
    delay 0.0;
  end loop;
end TUBE_MANAGER;

-------------------------------------------------------------------------
-- Procedure RESET_BOILER_PREVIOUS_VALUES sets the values
-- of "prev_" fields to the value of the corresponding non-"prev_" field.
--  See BOILER record specification for further clarification          --
-------------------------------------------------------------------------

procedure RESET_BOILER_PREVIOUS_VALUES (BLR: in out BOILER) is

begin
  BLR.STEAM_DRUM.PREV_WATER_LEVEL := BLR.STEAM_DRUM.WATER_LEVEL;
  BLR.STEAM_DRUM.PREV_PRESSURE := BLR.STEAM_DRUM.PRESSURE;
  BLR.STEAM_DRUM.PREV_FLOW_IN := BLR.STEAM_DRUM.FLOW_IN;
  BLR.STEAM_DRUM.PREV_FLOW_OUT := BLR.STEAM_DRUM.FLOW_OUT;
  BLR.STEAM_DRUM.PREV_TEMP:= BLR.STEAM_DRUM.TEMP;
  BLR.WATER_DRUM.PREV_FLOW_IN := BLR.WATER_DRUM.FLOW_IN;
  BLR.WATER_DRUM.PREV_FLOW_OUT := BLR.WATER_DRUM.FLOW_OUT;
  BLR.SUPERHEATER.PREV_FLOW_IN := BLR.SUPERHEATER.FLOW_IN;
  BLR.SUPERHEATER.PREV_FLOW_OUT := BLR.SUPERHEATER.FLOW_OUT;
  BLR.SUPERHEATER.PREV_RUPTURE := BLR.SUPERHEATER.RUPTURE;
  BLR.SUPERHEATER.PREV_TEMP := BLR.SUPERHEATER.TEMP;
  BLR.DESUPERHEATER.PREV_FLOW_IN := BLR.DESUPERHEATER.FLOW_IN;
  BLR.DESUPERHEATER.PREV_FLOW_OUT := BLR.DESUPERHEATER.FLOW_OUT;
  BLR.DESUPERHEATER.PREV_RUPTURE := BLR.DESUPERHEATER.RUPTURE;
  BLR.DESUPERHEATER.PREV_TEMP := BLR.DESUPERHEATER.TEMP;
  BLR.GENERATION_TUBES.PREV_FLOW_IN := BLR.GENERATION_TUBES.FLOW_IN;
  BLR.GENERATION_TUBES.PREV_FLOW_OUT :=
                       BLR.GENERATION_TUBES.FLOW_OUT;
  BLR.GENERATION_TUBES.PREV_RUPTURE := BLR.GENERATION_TUBES.RUPTURE;
  BLR.GENERATION_TUBES.PREV_TEMP := BLR.GENERATION_TUBES.TEMP;
  BLR.FURNACE.PREV_DECK_STATUS := BLR.FURNACE.DECK_STATUS;
  BLR.FURNACE.PREV_FIRING_RATE := BLR.FURNACE.FIRING_RATE;
  BLR.FURNACE.PREV_EXPLOSION := BLR.FURNACE.EXPLOSION;
  BLR.FURNACE.PREV_PERISCOPE := BLR.FURNACE.PERISCOPE;
  BLR.FURNACE.PREV_FIRES_LIT := BLR.FURNACE.FIRES_LIT;
  BLR.FURNACE.PREV_FIRE_APPEARANCE := BLR.FURNACE.FIRE_APPEARANCE;
end RESET_BOILER_PREVIOUS_VALUES;

-------------------------------------------------------------------------
-- Procedure INITIALIZE_BOILER sets all boiler components to normal steady
-- state steaming values                                             --
-------------------------------------------------------------------------

procedure INITIALIZE_BOILER (BLR: in out BOILER) is

begin
  BLR.STEAM_DRUM.WATER_LEVEL:= NORM;
```

```
  BLR.STEAM_DRUM.PRESSURE:= NORM;
  BLR.STEAM_DRUM.FLOW_IN:= NORM;
  BLR.STEAM_DRUM.FLOW_OUT:= NORM;
  BLR.STEAM_DRUM.TEMP:= NORM;
  BLR.WATER_DRUM.FLOW_IN:= NORM;
  BLR.WATER_DRUM.FLOW_OUT:= NORM;
  BLR.SUPERHEATER.FLOW_IN:= NORM;
  BLR.SUPERHEATER.FLOW_OUT:= NORM;
  BLR.SUPERHEATER.RUPTURE:= FALSE;
  BLR.SUPERHEATER.TEMP:= NORM;
  BLR.SUPERHEATER.DIRECT_HEAT_CONTACT:= TRUE;
  BLR.DESUPERHEATER.FLOW_IN:= NORM;
  BLR.DESUPERHEATER.FLOW_OUT:= NORM;
  BLR.DESUPERHEATER.RUPTURE:= FALSE;
  BLR.DESUPERHEATER.TEMP:= NORM;
  BLR.DESUPERHEATER.DIRECT_HEAT_CONTACT:= FALSE;
  BLR.GENERATION_TUBES.FLOW_IN:= NORM;
  BLR.GENERATION_TUBES.FLOW_OUT:= NORM;
  BLR.GENERATION_TUBES.RUPTURE:= FALSE;
  BLR.GENERATION_TUBES.TEMP:= NORM;
  BLR.GENERATION_TUBES.DIRECT_HEAT_CONTACT:= TRUE;
  BLR.FURNACE.DECK_STATUS:= NO_FUEL_ON_DECK;
  BLR.FURNACE.FIRING_RATE:= NORM;
  BLR.FURNACE.EXPLOSION:= FALSE;
  BLR.FURNACE.PERISCOPE:= CLEAR;
  BLR.FURNACE.FIRES_LIT:= TRUE;
  BLR.FURNACE.FIRE_APPEARANCE:= FAN_SHAPED;
  RESET_BOILER_PREVIOUS_VALUES(BLR);
end INITIALIZE_BOILER;

-------------------------------------------------------------------------
-- Procedure DISPLAY_BOILER_STATUS shows the values of boiler parameters.
-- Some may not be measurable in real life, but their display here demonstrates more fully the
-- causal chain of events                    --
-------------------------------------------------------------------------

procedure DISPLAY_BOILER_STATUS(BLR: in out BOILER) is

begin
  if (BLR.STEAM_DRUM.WATER_LEVEL /=
              BLR.STEAM_DRUM.PREV_WATER_LEVEL or
                 BLR.STEAM_DRUM.PRESSURE /= BLR.STEAM_DRUM.PREV_PRESSURE or
              BLR.STEAM_DRUM.FLOW_IN /=
              BLR.STEAM_DRUM.PREV_FLOW_IN or
              BLR.STEAM_DRUM.FLOW_OUT /=
              BLR.STEAM_DRUM.PREV_FLOW_OUT or
              BLR.STEAM_DRUM.TEMP /= BLR.STEAM_DRUM.PREV_TEMP or
              BLR.WATER_DRUM.FLOW_IN /=
              BLR.WATER_DRUM.PREV_FLOW_IN or
              BLR.WATER_DRUM.FLOW_OUT /=
              BLR.WATER_DRUM.PREV_FLOW_OUT or
              BLR.SUPERHEATER.FLOW_IN /=
```

```
        BLR.SUPERHEATER.PREV_FLOW_IN or
        BLR.SUPERHEATER.FLOW_OUT /=
        BLR.SUPERHEATER.PREV_FLOW_OUT or
        BLR.SUPERHEATER.RUPTURE /=
        BLR.SUPERHEATER.PREV_RUPTURE or
        BLR.SUPERHEATER.TEMP /= BLR.SUPERHEATER.PREV_TEMP or
        BLR.DESUPERHEATER.FLOW_IN /=
        BLR.DESUPERHEATER.PREV_FLOW_IN or
        BLR.DESUPERHEATER.FLOW_OUT /=
        BLR.DESUPERHEATER.PREV_FLOW_OUT or
        BLR.DESUPERHEATER.RUPTURE /=
        BLR.DESUPERHEATER.PREV_RUPTURE or
        BLR.DESUPERHEATER.TEMP /= BLR.DESUPERHEATER.PREV_TEMP or
        BLR.GENERATION_TUBES.FLOW_IN /=
        BLR.GENERATION_TUBES.PREV_FLOW_IN or
        BLR.GENERATION_TUBES.FLOW_OUT /=
        BLR.GENERATION_TUBES.PREV_FLOW_OUT or
        BLR.GENERATION_TUBES.RUPTURE /=
        BLR.GENERATION_TUBES.PREV_RUPTURE or
        BLR.GENERATION_TUBES.TEMP /=
        BLR.GENERATION_TUBES.PREV_TEMP or
        BLR.FURNACE.DECK_STATUS /=
      BLR.FURNACE.PREV_DECK_STATUS or
      BLR.FURNACE.FIRING_RATE /= BLR.FURNACE.PREV_FIRING_RATE or
      BLR.FURNACE.EXPLOSION /= BLR.FURNACE.PREV_EXPLOSION or
      BLR.FURNACE.FIRES_LIT /= BLR.FURNACE.PREV_FIRES_LIT or
      BLR.FURNACE.PERISCOPE /= BLR.FURNACE.PREV_PERISCOPE or
      BLR.FURNACE.FIRE_APPEARANCE /=
      BLR.FURNACE.PREV_FIRE_APPEARANCE) then

NEW_LINE;
SET_COL(30);
PUT_LINE("PLANT STATUS--BOILER");
NEW_LINE;
PUT("BOILER STEAM DRUM WATER LEVEL: ");
PUT(BLR.STEAM_DRUM.WATER_LEVEL);
SET_COL(45);
PUT("BOILER STEAM DRUM PRESSURE: ");
PUT(BLR.STEAM_DRUM.PRESSURE);
NEW_LINE;
PUT("BOILER STEAM DRUM FLOW IN: ");
PUT(BLR.STEAM_DRUM.FLOW_IN);
SET_COL(45);
PUT("BOILER STEAM DRUM FLOW OUT: ");
PUT(BLR.STEAM_DRUM.FLOW_OUT);
NEW_LINE;
PUT("BOILER STEAM DRUM TEMP: ");
PUT(BLR.STEAM_DRUM.TEMP);
NEW_LINE(2);
PUT("BOILER WATER DRUM FLOW IN: ");
PUT(BLR.WATER_DRUM.FLOW_IN);
SET_COL(45);
```

```
PUT("BOILER WATER DRUM FLOW OUT: ");
PUT(BLR.WATER_DRUM.FLOW_OUT);
NEW_LINE(2);
PUT("SUPERHEATER FLOW IN: ");
PUT(BLR.SUPERHEATER.FLOW_IN);
SET_COL(45);
PUT("SUPERHEATER FLOW OUT: ");
PUT(BLR.SUPERHEATER.FLOW_OUT);
NEW_LINE;
PUT("SUPERHEATER RUPTURE: ");
PUT(BLR.SUPERHEATER.RUPTURE);
SET_COL(45);
PUT("SUPERHEATER TEMP: ");
PUT(BLR.SUPERHEATER.TEMP);
NEW_LINE(2);
PUT("DESUPERHEATER FLOW IN: ");
PUT(BLR.DESUPERHEATER.FLOW_IN);
SET_COL(45);
PUT("DESUPERHEATER FLOW OUT: ");
PUT(BLR.DESUPERHEATER.FLOW_OUT);
NEW_LINE;
PUT("DESUPERHEATER RUPTURE: ");
PUT(BLR.DESUPERHEATER.RUPTURE);
SET_COL(45);
PUT("DESUPERHEATER TEMP: ");
PUT(BLR.DESUPERHEATER.TEMP);
NEW_LINE(2);
PUT("GENERATION TUBES FLOW IN: ");
PUT(BLR.GENERATION_TUBES.FLOW_IN);
SET_COL(45);
PUT("GENERATION TUBES FLOW OUT: ");
PUT(BLR.GENERATION_TUBES.FLOW_OUT);
NEW_LINE;
PUT("GENERATION TUBES RUPTURE: ");
PUT(BLR.GENERATION_TUBES.RUPTURE);
SET_COL(45);
PUT("GENERATION TUBES TEMP: ");
PUT(BLR.GENERATION_TUBES.TEMP);
NEW_LINE(2);
PUT("FURNACE DECK STATUS: ");
PUT(BLR.FURNACE.DECK_STATUS);
SET_COL(45);
PUT("FIRING RATE: ");
PUT(BLR.FURNACE.FIRING_RATE);
NEW_LINE;
PUT("BOILER EXPLOSION: ");
PUT(BLR.FURNACE.EXPLOSION);
SET_COL(45);
PUT("PERISCOPE: ");
PUT(BLR.FURNACE.PERISCOPE);
NEW_LINE;
PUT("FIRES LIT: ");
```

```
    PUT(BLR.FURNACE.FIRES_LIT);
    SET_COL(45);
    PUT("FIRE APPEARANCE: ");
    PUT(BLR.FURNACE.FIRE_APPEARANCE);
    NEW_LINE;
    for i in 1..80 loop
      PUT('-');
    end loop;
    NEW_LINE;
    for i in 1..80 loop
      PUT('/');
    end loop;
    NEW_LINE;
    for i in 1..80 loop
      PUT('-');
    end loop;
    NEW_LINE(2);
    RESET_BOILER_PREVIOUS_VALUES(BLR);
  end if;
end DISPLAY_BOILER_STATUS;

end BOILER_MODEL;
```

```
with TEXT_IO, VALVES_AND_PIPING, BOILER_MODEL, SYSTEM;
use TEXT_IO, VALVES_AND_PIPING, BOILER_MODEL, SYSTEM;

procedure MAIN is
  VALVE_OF_INTEREST: INDEX;
  CURRENT: VALVE_PTR:= VIRTUAL_SH_OUTLET;
  CHOICE, NEXT_CHOICE, LAST_CHOICE: CHARACTER;
  pragma PRIORITY(2);
  package INDEX_IO is new ENUMERATION_IO(INDEX);
  use INDEX_IO;

begin
  FILL_SYSTEM_ARRAYS;
  ASSIGN_VALVE_ORDER;
  INITIALIZE_BOILER(ALPHA_BOILER);
  INITIALIZE_PLANT;
  TRACE:= TRUE;
  PUT("CHANGE VALVE (C)HARACTERISTIC OR (V)ALVE STATUS?");
  GET(CHOICE);
  NEW_LINE;
  PUT("ENTER VALVE TO CHANGE: ");
  GET(VALVE_OF_INTEREST);
  while CURRENT.VALVE_ID /= VALVE_OF_INTEREST loop
    CURRENT:= CURRENT.NEXT;
  end loop;
  if CHOICE = 'C' or CHOICE = 'c' then
    PUT("CHANGE INPUT (F)LOW OR INPUT (P)RESSURE?");
    GET(NEXT_CHOICE);
    NEW_LINE;
    PUT("CHANGE TO (N)ONE, (L)OW, N(O)RM, (H)IGH?");
    GET(LAST_CHOICE);
    NEW_LINE;
    if NEXT_CHOICE = 'F' or NEXT_CHOICE = 'f' then
      case LAST_CHOICE is
        when 'N' | 'n' => CURRENT.INPUT_FLOW:= NONE;
        when 'L' | 'l' => CURRENT.INPUT_FLOW:= LOW;
        when 'O' | 'o' => CURRENT.INPUT_FLOW:= NORM;
        when 'H' | 'h' => CURRENT.INPUT_FLOW:= HIGH;
        when others => null;
      end case;
    elsif NEXT_CHOICE = 'P' or NEXT_CHOICE = 'p' then
      case LAST_CHOICE is
        when 'N' | 'n' => CURRENT.INPUT_PRESSURE:= NONE;
        when 'L' | 'l' => CURRENT.INPUT_PRESSURE:= LOW;
        when 'O' | 'o' => CURRENT.INPUT_PRESSURE:= NORM;
        when 'H' | 'h' => CURRENT.INPUT_PRESSURE:= HIGH;
        when others => null;
      end case;
    end if;
  elsif CHOICE = 'V' or CHOICE = 'v' then
    if CURRENT.STATUS = OPEN then
      CURRENT.STATUS:= SHUT;
    else
      CURRENT.STATUS:= OPEN;
    end if;
```

```
  end if;
  loop
    DISPLAY_BOILER_STATUS(ALPHA_BOILER);
    delay 0.5;
  end loop;
end MAIN;
```

APPENDIX B (BoilerModel TEST RESULTS)

```
CHANGE VALVE (C)HARACTERISTIC OR (V)ALVE STATUS?v

ENTER VALVE TO CHANGE: fwcv

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------- ----------- ------------- -----------
FWCV       SHUT   NORM          NORM        NONE          NONE

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------- ----------- ------------- -----------
FEED_STOP  OPEN   HIGH          NORM        HIGH          NORM

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------- ----------- ------------- -----------
FWCV       SHUT   HIGH          NORM        NONE          NONE

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------- ----------- ------------- -----------
MAN_CHK    OPEN   NONE          NONE        NONE          NONE

                PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM      BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NONE          BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE          BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NORM                SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE               SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM              DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE             DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE           GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: FALSE          GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK     FIRING RATE: NORM
BOILER EXPLOSION: FALSE                  PERISCOPE: CLEAR
FIRES LIT: TRUE                          FIRE APPEARANCE: FAN_SHAPED
-------------------------------------------------------------------------------
///////////////////////////////////////////////////////////////////////////////
-------------------------------------------------------------------------------
```

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NONE BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW_ALARM BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NONE BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	SHUT	LOW	NONE	NONE	NONE

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW_ALARM   BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NONE   BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE   BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NONE   SUPERHEATER FLOW OUT: NONE
SUPERHEATER RUPTURE: FALSE   SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM   DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE   DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE   GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: TRUE   GENERATION TUBES TEMP: HIGH

FURNACE DECK STATUS: NO_FUEL_ON_DECK   FIRING RATE: NORM
BOILER EXPLOSION: FALSE   PERISCOPE: FOGGED
FIRES LIT: TRUE   FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MSS	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
DESUP_IN	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	LOW	NORM	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ER_BLKHD _STOP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AMR_BLKHD _STOP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MFP_STM _SUP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ONE_FIFTY _IN	OPEN	LOW	NORM	LOW	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW_ALARM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NONE BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NONE SUPERHEATER FLOW OUT: NONE
SUPERHEATER RUPTURE: TRUE SUPERHEATER TEMP: HIGH

DESUPERHEATER FLOW IN: NONE DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: TRUE GENERATION TUBES TEMP: HIGH

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: FOGGED
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MSS	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
DESUP_IN	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	LOW	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ONE_FIFTY _IN	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ER_BLKHD _STOP	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AMR_BLKHD _STOP	OPEN	LOW	NONE	LOW	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MFP_STM_ SUP	OPEN	LOW	NONE	LOW	NONE

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW_ALARM BOILER STEAM DRUM PRESSURE: NONE
BOILER STEAM DRUM FLOW IN: NONE BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NONE SUPERHEATER FLOW OUT: NONE
SUPERHEATER RUPTURE: TRUE SUPERHEATER TEMP: HIGH

DESUPERHEATER FLOW IN: NONE DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NONE GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: TRUE GENERATION TUBES TEMP: HIGH

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: FOGGED
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: LOW_ALARM BOILER STEAM DRUM PRESSURE: NONE
BOILER STEAM DRUM FLOW IN: NONE BOILER STEAM DRUM FLOW OUT: NONE
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NONE BOILER WATER DRUM FLOW OUT: NONE

SUPERHEATER FLOW IN: NONE SUPERHEATER FLOW OUT: NONE
SUPERHEATER RUPTURE: TRUE SUPERHEATER TEMP: HIGH

DESUPERHEATER FLOW IN: NONE DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: HIGH

GENERATION TUBES FLOW IN: NONE GENERATION TUBES FLOW OUT: NONE
GENERATION TUBES RUPTURE: TRUE GENERATION TUBES TEMP: HIGH

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: FOGGED
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MSS	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
DESUP_IN	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	SHUT	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ER_BLKHD _STOP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AMR_BLKHD _STOP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MFP_STM _SUP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ONE_FIFTY _IN	OPEN	NONE	NONE	NONE	NONE

Oh, master Jim, how may I serve you? main
CHANGE VALVE (C)HARACTERISTIC OR (V)ALVE STATUS?c

ENTER VALVE TO CHANGE: fosp_disch
CHANGE INPUT (F)LOW OR INPUT (P)RESSURE?p

CHANGE TO (N)ONE, (L)OW, N(O)RM, (H)IGH?l

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FOSP_DISCH	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FOCV	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FO_RECIRC	SHUT	LOW	NORM	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FUEL_MAN	OPEN	LOW	NORM	LOW	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: TRUE FIRE APPEARANCE: IRREGULAR

//

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NONE
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: FALSE FIRE APPEARANCE: NONE

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MSS	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
DESUP_IN	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	SHUT	LOW	NORM	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	LOW	NORM	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ER_BLKHD_STOP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AMR_BLKHD_STOP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MFP_STM_SUP	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	LOW	NORM	LOW	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: FUEL_ON_DECK FIRING RATE: NONE
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: FALSE FIRE APPEARANCE: NONE

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	LOW	NORM	LOW	NORM

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ONE_FIFTY _IN	OPEN	LOW	NORM	LOW	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: LOW
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: LOW
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: FUEL_ON_DECK FIRING RATE: NONE
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: FALSE FIRE APPEARANCE: NONE

//

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: LOW
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: LOW SUPERHEATER FLOW OUT: LOW
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: LOW
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: FUEL_ON_DECK FIRING RATE: NONE
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: FALSE FIRE APPEARANCE: NONE

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	SHUT	LOW	LOW	NONE	NONE

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: LOW
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: LOW SUPERHEATER FLOW OUT: LOW
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: LOW DESUPERHEATER FLOW OUT: LOW
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: LOW
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: FUEL_ON_DECK FIRING RATE: NONE
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: FALSE FIRE APPEARANCE: NONE

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	OPEN	LOW	LOW	LOW	LOW

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MSS	OPEN	LOW	LOW	LOW	LOW

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	LOW	LOW	LOW	LOW

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
DESUP_IN	OPEN	LOW	LOW	LOW	LOW

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	LOW	LOW	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	LOW	LOW	LOW	LOW

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	LOW	LOW	LOW	LOW

```
VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
ONE_FIFTY
_IN        OPEN   LOW           LOW         LOW           LOW

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
ER_BLKHD_
STOP       OPEN   LOW           LOW         LOW           LOW

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
AMR_BLKHD
_STOP      OPEN   LOW           LOW         LOW           LOW

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
MFP_STM
_SUP       OPEN   LOW           LOW         LOW           LOW

                PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: HIGH      BOILER STEAM DRUM PRESSURE: LOW
BOILER STEAM DRUM FLOW IN: NORM          BOILER STEAM DRUM FLOW OUT: LOW
BOILER STEAM DRUM TEMP: LOW

BOILER WATER DRUM FLOW IN: NORM          BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: LOW                 SUPERHEATER FLOW OUT: LOW
SUPERHEATER RUPTURE: FALSE               SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: LOW               DESUPERHEATER FLOW OUT: LOW
DESUPERHEATER RUPTURE: FALSE             DESUPERHEATER TEMP: LOW

GENERATION TUBES FLOW IN: NORM           GENERATION TUBES FLOW OUT: LOW
GENERATION TUBES RUPTURE: FALSE          GENERATION TUBES TEMP: LOW

FURNACE DECK STATUS: FUEL_ON_DECK        FIRING RATE: NONE
BOILER EXPLOSION: FALSE                  PERISCOPE: CLEAR
FIRES LIT: FALSE                         FIRE APPEARANCE: NONE
-------------------------------------------------------------------------------
///////////////////////////////////////////////////////////////////////////////
-------------------------------------------------------------------------------

Oh, master Jim, how may I serve you? main
CHANGE VALVE (C)HARACTERISTIC OR (V)ALVE STATUS?v

ENTER VALVE TO CHANGE: vsh

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
VSH        SHUT   NORM          NORM        NONE          NONE

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
VSH        SHUT   HIGH          NORM        NONE          NONE

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
MSS        OPEN   NONE          NONE        NONE          NONE

VALVE      STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------  ------ ------------  ----------  ------------  -----------
DESUP_IN   OPEN   NONE          NONE        NONE          NONE
```

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
BLEED	SHUT	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ER_BLKHD _STOP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AMR_BLKHD _STOP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
MFP_STM _SUP	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ASS	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
AUG	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
FDB_IN	OPEN	NONE	NONE	NONE	NONE

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: LIFT_SAFE_HI
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: CLEAR
FIRES LIT: TRUE FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
ONE_FIFTY _IN	OPEN	NONE	NONE	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	OPEN	NORM	NORM	NORM	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM   BOILER STEAM DRUM PRESSURE: LIFT_SAFE_HI
BOILER STEAM DRUM FLOW IN: NORM   BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM   BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM   SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE   SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE   DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE   DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM   GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE   GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK   FIRING RATE: NORM
BOILER EXPLOSION: FALSE   PERISCOPE: CLEAR
FIRES LIT: TRUE   FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	SHUT	LIFT_SAFE_HI	NORM	NONE	NONE

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	OPEN	LIFT_SAFE_HI	NORM	LIFT_SAFE_HI	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM   BOILER STEAM DRUM PRESSURE: HIGH
BOILER STEAM DRUM FLOW IN: NORM   BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM   BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM   SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE   SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE   DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE   DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM   GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE   GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK   FIRING RATE: NORM
BOILER EXPLOSION: FALSE   PERISCOPE: CLEAR
FIRES LIT: TRUE   FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	OPEN	HIGH	NORM	HIGH	NORM

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM   BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM   BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM   BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM   SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE   SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE   DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE   DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM   GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE   GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK   FIRING RATE: NORM
BOILER EXPLOSION: FALSE   PERISCOPE: CLEAR
FIRES LIT: TRUE   FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
SAFETIES	SHUT	NORM	NORM	NONE	NONE

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM   BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM   BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM   BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM   SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE   SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE   DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE   DESUPERHEATER TEMP: HIGH

GENERATION TUBES FLOW IN: NORM   GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE   GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK   FIRING RATE: NORM
BOILER EXPLOSION: FALSE   PERISCOPE: CLEAR
FIRES LIT: TRUE   FIRE APPEARANCE: FAN_SHAPED

//

VALVE	STATUS	INPUT PRESS	INPUT FLOW	OUTPUT PRESS	OUTPUT FLOW
VSH	SHUT	NORM	NORM	NONE	NONE

```
VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------- ----------- ------------- -----------
VSH         SHUT   HIGH          NORM        NONE          NONE

VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------- ----------- ------------- -----------
SAFETIES    OPEN   NORM          NORM        NORM          NORM

                PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM      BOILER STEAM DRUM PRESSURE: LIFT_SAFE_HI
BOILER STEAM DRUM FLOW IN: NORM          BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM          BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM                SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE               SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE              DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE             DESUPERHEATER TEMP: HIGH

GENERATION TUBES FLOW IN: NORM           GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE          GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK     FIRING RATE: NORM
BOILER EXPLOSION: FALSE                  PERISCOPE: CLEAR
FIRES LIT: TRUE                          FIRE APPEARANCE: FAN_SHAPED
-------------------------------------------------------------------------------------------------------------------------
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
-------------------------------------------------------------------------------------------------------------------------


VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------- ----------- ------------- -----------
VSH         SHUT   LIFT_SAFE_HI  NORM        NONE          NONE

VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------- ----------- ------------- -----------
SAFETIES    OPEN   LIFT_SAFE_HI  NORM        LIFT_SAFE_HI  NORM

VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------- ----------- ------------- -----------
SAFETIES    OPEN   HIGH          NORM        HIGH          NORM

                PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM      BOILER STEAM DRUM PRESSURE: HIGH
BOILER STEAM DRUM FLOW IN: NORM          BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM          BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM                SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE               SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE              DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE             DESUPERHEATER TEMP: HIGH

GENERATION TUBES FLOW IN: NORM           GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE          GENERATION TUBES TEMP: NORM
```

```
FURNACE DECK STATUS: NO_FUEL_ON_DECK      FIRING RATE: NORM
BOILER EXPLOSION: FALSE                   PERISCOPE: CLEAR
FIRES LIT: TRUE                           FIRE APPEARANCE: FAN_SHAPED
-------------------------------------------------------------------------------------------------------------------
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
-------------------------------------------------------------------------------------------------------------------


                    PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM       BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM           BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM           BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM                 SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE                SUPERHEATER TEMP: NORM

DESUPERHEATER FLOW IN: NONE               DESUPERHEATER FLOW OUT: NONE
DESUPERHEATER RUPTURE: FALSE              DESUPERHEATER TEMP: HIGH

GENERATION TUBES FLOW IN: NORM            GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE           GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK      FIRING RATE: NORM
BOILER EXPLOSION: FALSE                   PERISCOPE: CLEAR
FIRES LIT: TRUE                           FIRE APPEARANCE: FAN_SHAPED
-------------------------------------------------------------------------------------------------------------------
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
-------------------------------------------------------------------------------------------------------------------


VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------  ----------  ------------  -----------
SAFETIES    SHUT   NORM          NORM        NONE          NONE

Oh, master Jim, how may I serve you? main
CHANGE VALVE (C)HARACTERISTIC OR (V)ALVE STATUS?c

ENTER VALVE TO CHANGE: air_reg
CHANGE INPUT (F)LOW OR INPUT (P)RESSURE?f

CHANGE TO (N)ONE, (L)OW, N(O)RM, (H)IGH?h

VALVE       STATUS INPUT PRESS   INPUT FLOW  OUTPUT PRESS  OUTPUT FLOW
---------   ------ ------------  ----------  ------------  -----------
AIR_REG     OPEN   NORM          HIGH        NORM          HIGH

                    PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM       BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM           BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM           BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM                 SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE                SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM               DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE              DESUPERHEATER TEMP: NORM
```

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: FALSE PERISCOPE: ORANGE
FIRES LIT: TRUE FIRE APPEARANCE: IRREGULAR

//

PLANT STATUS--BOILER

BOILER STEAM DRUM WATER LEVEL: NORM BOILER STEAM DRUM PRESSURE: NORM
BOILER STEAM DRUM FLOW IN: NORM BOILER STEAM DRUM FLOW OUT: NORM
BOILER STEAM DRUM TEMP: NORM

BOILER WATER DRUM FLOW IN: NORM BOILER WATER DRUM FLOW OUT: NORM

SUPERHEATER FLOW IN: NORM SUPERHEATER FLOW OUT: NORM
SUPERHEATER RUPTURE: FALSE SUPERHEATER TEMP: LOW

DESUPERHEATER FLOW IN: NORM DESUPERHEATER FLOW OUT: NORM
DESUPERHEATER RUPTURE: FALSE DESUPERHEATER TEMP: NORM

GENERATION TUBES FLOW IN: NORM GENERATION TUBES FLOW OUT: NORM
GENERATION TUBES RUPTURE: FALSE GENERATION TUBES TEMP: NORM

FURNACE DECK STATUS: NO_FUEL_ON_DECK FIRING RATE: NORM
BOILER EXPLOSION: TRUE PERISCOPE: ORANGE
FIRES LIT: TRUE FIRE APPEARANCE: IRREGULAR

//

APPENDIX C (ADA/LISP COMPARISON CODE)

ADA CODE

```
with TEXT_IO;

use TEXT_IO;

procedure BOILER_MODEL is


 type MEASUREMENT_VALUE is (OPEN, SHUT, HIGH, NORM, LOW, NONE,
                   HIGH_BUT_NOT_ALARM, LOW_BUT_NOT_ALARM, HIGH_ALARM,
                 LOW_ALARM, SAFETIES_LIFT_HIGH, DRAG_OFF_LINE_LOW,
                 CLEAR, ORANGE);

 type INDEX is (A_MSS, A_ASS, A_DESUP_IN, A_SH_PROT_IN, A_SAFETIES,
                 A_AIR_REG,  A_SHUTTERS, A_FUEL_MAN, A_FOCV,
                 A_FO_RECIRC, FUEL_PUMP_DISCH, A_FEED_STOP, A_FWCV,
                 A_MAN_CHK, A_BLEED, AUG, FDB_IN, ONE_FIFTY_IN,
                 SOOT_BLOW_SUP, PMAC_IN, ER_BLKHD_STOP,
                 AMR_BLKHD_STOP, MFP_STM_SUP);

 package IIO is new INTEGER_IO(INTEGER);
 use IIO;
 package INDEX_IO is new ENUMERATION_IO(INDEX);
 use INDEX_IO;

 package MEASUREMENT_VALUE_IO is new
                     ENUMERATION_IO(MEASUREMENT_VALUE);
 use MEASUREMENT_VALUE_IO;

 type VALVE;


 type VALVE_PTR is access VALVE;

 type CROSS_CHECK is array(A_MSS..MFP_STM_SUP) of VALVE_PTR;
 VALVE_LOOK_UP: CROSS_CHECK;

 type VALVE_PTR_ARRAY is array(1..10) of VALVE_PTR;

 type VALVE is
  record
    VALVE_ID: INDEX;
    UPSTREAM, DOWNSTREAM: VALVE_PTR_ARRAY;
    COUNTED: BOOLEAN:= FALSE;
    NEXT: VALVE_PTR;
```

```
    STATUS: MEASUREMENT_VALUE:= SHUT;
    INPUT: MEASUREMENT_VALUE:= NONE;
    OUTPUT: MEASUREMENT_VALUE:= NONE;
   end record;

  VIRTUAL_SH_OUTLET: VALVE_PTR:= new VALVE;
  ALPHA_MAIN_STEAM_STOP: VALVE_PTR:= new VALVE;
  ALPHA_AUX_STEAM_STOP: VALVE_PTR:= new VALVE;
  ALPHA_DESUPERHEATER_INLET: VALVE_PTR:= new VALVE;
  ALPHA_SUPERHEATER_PROTECTION_INLET: VALVE_PTR:= new VALVE;
  ALPHA_SAFETIES: VALVE_PTR:= new VALVE;
  ALPHA_AIR_REGISTERS: VALVE_PTR:= new VALVE;
  ALPHA_AIR_SHUTTERS: VALVE_PTR:= new VALVE;
  ALPHA_FUEL_MANIFOLD: VALVE_PTR:= new VALVE;
  ALPHA_FUEL_OIL_CONTROL_VALVE: VALVE_PTR:= new VALVE;
  ALPHA_FUEL_RECIRC: VALVE_PTR:= new VALVE;
  FO_SVC_PUMP_DISCH: VALVE_PTR:= new VALVE;
  ALPHA_FEED_STOP_VALVE: VALVE_PTR:= new VALVE;
  ALPHA_FWCV: VALVE_PTR:= new VALVE;
  ALPHA_MANUAL_CHECK_VLV: VALVE_PTR:= new VALVE;
  ALPHA_BLEEDER: VALVE_PTR:= new VALVE;
  AUGMENTOR: VALVE_PTR:= new VALVE;
  FDB_INLET: VALVE_PTR:= new VALVE;
  ONE_FIFTY_INLET: VALVE_PTR:= new VALVE;
  SOOT_BLOWER_SUPPLY: VALVE_PTR:= new VALVE;
  PMAC_INLET: VALVE_PTR:= new VALVE;
  ER_BULKHEAD_STOP: VALVE_PTR:= new VALVE;
  AMR_BULKHEAD_STOP: VALVE_PTR:= new VALVE;
  MFP_STM_SUPPLY: VALVE_PTR:= new VALVE;


  TRACE, REVISED, REPROPAGATE, GONE_THRU: boolean:= FALSE;

  type BOILER is
   record
    WATER_LEVEL: MEASUREMENT_VALUE:= NORM;
    STEAM_DRUM_PRESSURE, SUPERHEATER_OUTLET_TEMP:
        MEASUREMENT_VALUE:= LOW;
    FUEL_PRESSURE, AIR: MEASUREMENT_VALUE:= NONE;
    FIRE_STATUS: BOOLEAN:= FALSE;
   end record;


  ALPHA_BOILER: BOILER;

--procedure to establish valve upstream/downstream order

  procedure ASSIGN_VALVE_ORDER is

  begin
   VIRTUAL_SH_OUTLET.DOWNSTREAM(1):= ALPHA_MAIN_STEAM_STOP;
   VIRTUAL_SH_OUTLET.DOWNSTREAM(2):= ALPHA_DESUPERHEATER_INLET;
   ALPHA_MAIN_STEAM_STOP.UPSTREAM(1):= VIRTUAL_SH_OUTLET;
   ALPHA_MAIN_STEAM_STOP.DOWNSTREAM(1):= ER_BULKHEAD_STOP;
   ALPHA_MAIN_STEAM_STOP.DOWNSTREAM(2):= AMR_BULKHEAD_STOP;
   ALPHA_MAIN_STEAM_STOP.DOWNSTREAM(3):= MFP_STM_SUPPLY;
   ALPHA_DESUPERHEATER_INLET.UPSTREAM(1):= VIRTUAL_SH_OUTLET;
    ALPHA_DESUPERHEATER_INLET.DOWNSTREAM(1):= ALPHA_AUX_STEAM_STOP;
```

```
  ALPHA_DESUPERHEATER_INLET.DOWNSTREAM(2):= ALPHA_BLEEDER;
  ALPHA_AUX_STEAM_STOP.UPSTREAM(1):= ALPHA_DESUPERHEATER_INLET;
  ALPHA_AUX_STEAM_STOP.DOWNSTREAM(1):= AUGMENTOR;
  ALPHA_AUX_STEAM_STOP.DOWNSTREAM(2):= FDB_INLET;
  ALPHA_AUX_STEAM_STOP.DOWNSTREAM(3):= ONE_FIFTY_INLET;
  ALPHA_AUX_STEAM_STOP.DOWNSTREAM(4):= SOOT_BLOWER_SUPPLY;
  ALPHA_AUX_STEAM_STOP.DOWNSTREAM(5):= PMAC_INLET;
  ALPHA_AIR_REGISTERS.UPSTREAM(1):= ALPHA_AIR_SHUTTERS;
  ALPHA_AIR_SHUTTERS.DOWNSTREAM(1):= ALPHA_AIR_REGISTERS;
   ALPHA_FUEL_MANIFOLD.UPSTREAM(1):= ALPHA_FUEL_OIL_CONTROL_VALVE;
  ALPHA_FUEL_OIL_CONTROL_VALVE.UPSTREAM(1):= FO_SVC_PUMP_DISCH;
  ALPHA_FUEL_OIL_CONTROL_VALVE.DOWNSTREAM(1):=
       ALPHA_FUEL_MANIFOLD;
   ALPHA_FUEL_OIL_CONTROL_VALVE.DOWNSTREAM(2):= ALPHA_FUEL_RECIRC;
  ALPHA_FUEL_RECIRC.UPSTREAM(1):= ALPHA_FUEL_OIL_CONTROL_VALVE;
  ALPHA_FUEL_RECIRC.UPSTREAM(2):= ALPHA_FUEL_MANIFOLD;
   FO_SVC_PUMP_DISCH.DOWNSTREAM(1):= ALPHA_FUEL_OIL_CONTROL_VALVE;
  ALPHA_FEED_STOP_VALVE.DOWNSTREAM(1):= ALPHA_FWCV;
  ALPHA_FWCV.UPSTREAM(1):= ALPHA_FEED_STOP_VALVE;
  ALPHA_FWCV.DOWNSTREAM(1):= ALPHA_MANUAL_CHECK_VLV;
  ALPHA_MANUAL_CHECK_VLV.UPSTREAM(1):= ALPHA_FWCV;
  ALPHA_BLEEDER.UPSTREAM(1):= ALPHA_DESUPERHEATER_INLET;
  AUGMENTOR.UPSTREAM(1):= ALPHA_AUX_STEAM_STOP;
  FDB_INLET.UPSTREAM(1):= ALPHA_AUX_STEAM_STOP;
  ONE_FIFTY_INLET.UPSTREAM(1):= ALPHA_AUX_STEAM_STOP;
  SOOT_BLOWER_SUPPLY.UPSTREAM(1):= ALPHA_AUX_STEAM_STOP;
  PMAC_INLET.UPSTREAM(1):= ALPHA_AUX_STEAM_STOP;
  ER_BULKHEAD_STOP.UPSTREAM(1):= ALPHA_MAIN_STEAM_STOP;
  AMR_BULKHEAD_STOP.UPSTREAM(1):= ALPHA_MAIN_STEAM_STOP;
  MFP_STM_SUPPLY.UPSTREAM(1):= ALPHA_MAIN_STEAM_STOP;

 end ASSIGN_VALVE_ORDER;


--procedure to "start up" a boiler

 procedure LIGHT_OFF is

 begin

  ALPHA_BOILER.STEAM_DRUM_PRESSURE:= NORM;
  ALPHA_BOILER.SUPERHEATER_OUTLET_TEMP:= NORM;
  ALPHA_BOILER.FUEL_PRESSURE:= NORM;
  ALPHA_BOILER.AIR:= NORM;
  ALPHA_BOILER.FIRE_STATUS:= TRUE;

 end LIGHT_OFF;

--procedure to put boiler on the line and align valves

 procedure INITIALIZE_PLANT is

  CURRENT: VALVE_PTR:= ALPHA_MAIN_STEAM_STOP;

 begin

  ASSIGN_VALVE_ORDER;
  LIGHT_OFF;
```

```
  VIRTUAL_SH_OUTLET.STATUS:= OPEN;
  VIRTUAL_SH_OUTLET.INPUT:= NORM;
  ALPHA_MAIN_STEAM_STOP.STATUS:= OPEN;
  ALPHA_MAIN_STEAM_STOP.NEXT:= ALPHA_AUX_STEAM_STOP;
  ALPHA_AUX_STEAM_STOP.STATUS:= OPEN;
  ALPHA_AUX_STEAM_STOP.NEXT:= ALPHA_DESUPERHEATER_INLET;
  ALPHA_DESUPERHEATER_INLET.STATUS:= OPEN;
  ALPHA_DESUPERHEATER_INLET.NEXT:=
              ALPHA_SUPERHEATER_PROTECTION_INLET;
  ALPHA_SUPERHEATER_PROTECTION_INLET.STATUS:= SHUT;
  ALPHA_SUPERHEATER_PROTECTION_INLET.NEXT:= ALPHA_SAFETIES;
  ALPHA_SUPERHEATER_PROTECTION_INLET.INPUT:= NORM;
  ALPHA_SAFETIES.STATUS:= SHUT;
  ALPHA_SAFETIES.NEXT:= ALPHA_AIR_REGISTERS;
  ALPHA_SAFETIES.INPUT:= NORM;
  ALPHA_AIR_REGISTERS.STATUS:= OPEN;
  ALPHA_AIR_REGISTERS.NEXT:= ALPHA_AIR_SHUTTERS;
  ALPHA_AIR_SHUTTERS.STATUS:= OPEN;
  ALPHA_AIR_SHUTTERS.NEXT:= ALPHA_FUEL_MANIFOLD;
  ALPHA_AIR_SHUTTERS.INPUT:= NORM;
  ALPHA_FUEL_MANIFOLD.STATUS:= OPEN;
  ALPHA_FUEL_MANIFOLD.NEXT:= ALPHA_FUEL_OIL_CONTROL_VALVE;
  ALPHA_FUEL_OIL_CONTROL_VALVE.STATUS:= OPEN;
  ALPHA_FUEL_OIL_CONTROL_VALVE.NEXT:= ALPHA_FUEL_RECIRC;
  ALPHA_FUEL_RECIRC.STATUS:= SHUT;
  ALPHA_FUEL_RECIRC.NEXT:= FO_SVC_PUMP_DISCH;
  FO_SVC_PUMP_DISCH.STATUS:= OPEN;
  FO_SVC_PUMP_DISCH.NEXT:= ALPHA_FEED_STOP_VALVE;
  FO_SVC_PUMP_DISCH.INPUT:= NORM;
  ALPHA_FEED_STOP_VALVE.STATUS:= OPEN;
  ALPHA_FEED_STOP_VALVE.NEXT:= ALPHA_FWCV;
  ALPHA_FEED_STOP_VALVE.INPUT:= NORM;
  ALPHA_FWCV.STATUS:= OPEN;
  ALPHA_FWCV.NEXT:= ALPHA_MANUAL_CHECK_VLV;
  ALPHA_MANUAL_CHECK_VLV.STATUS:= OPEN;
  ALPHA_MANUAL_CHECK_VLV.NEXT:= ALPHA_BLEEDER;
  ALPHA_BLEEDER.STATUS:= SHUT;
  ALPHA_BLEEDER.NEXT:= AUGMENTOR;
  AUGMENTOR.STATUS:= OPEN;
  AUGMENTOR.NEXT:= FDB_INLET;
  FDB_INLET.STATUS:= OPEN;
  FDB_INLET.NEXT:= ONE_FIFTY_INLET;
  ONE_FIFTY_INLET.STATUS:= OPEN;
  ONE_FIFTY_INLET.NEXT:= SOOT_BLOWER_SUPPLY;
  SOOT_BLOWER_SUPPLY.STATUS:= OPEN;
  SOOT_BLOWER_SUPPLY.NEXT:= PMAC_INLET;
  PMAC_INLET.STATUS:= SHUT;
  PMAC_INLET.NEXT:= ER_BULKHEAD_STOP;
  ER_BULKHEAD_STOP.STATUS:= OPEN;
  ER_BULKHEAD_STOP.NEXT:= AMR_BULKHEAD_STOP;
  AMR_BULKHEAD_STOP.STATUS:= OPEN;
  AMR_BULKHEAD_STOP.NEXT:= MFP_STM_SUPPLY;
  MFP_STM_SUPPLY.STATUS:= OPEN;
  for i in A_MSS..MFP_STM_SUP loop
    CURRENT.VALVE_ID:= i;
    VALVE_LOOK_UP(i):= CURRENT;
    CURRENT:= CURRENT.NEXT;
  end loop;
end INITIALIZE_PLANT;
```

```
--procedure to provide trace to user

 procedure SHOW_VALVE_STATUS (AFFECTED: in out VALVE_PTR) is

  CURRENT: VALVE_PTR:= ALPHA_MAIN_STEAM_STOP;

 begin

  PUT("VALVE");
  SET_COL(30);
  PUT("STATUS");
  SET_COL(40);
  PUT("INPUT");
  SET_COL(60);
  PUT_LINE("OUTPUT");
  PUT("-----");
  SET_COL(30);
  PUT("------");
  SET_COL(40);
  PUT("-----");
  SET_COL(60);
  PUT("------");
  NEW_LINE(2);
  while CURRENT /= null loop
    PUT(CURRENT.VALVE_ID);
    SET_COL(30);
    PUT(CURRENT.STATUS);
    SET_COL(40);
    PUT(CURRENT.INPUT);
    SET_COL(60);
    PUT(CURRENT.OUTPUT);
    if CURRENT = AFFECTED then
      SET_COL(70);
      PUT("**CHANGE**");
    end if;
    NEW_LINE;
    CURRENT:= CURRENT.NEXT;
  end loop;
  PUT_LINE("-------------------------------------------------------------------------------");
  PUT_LINE("///////////////////////////////////////////////////////////////////////////////");
  PUT_LINE("-------------------------------------------------------------------------------");
  NEW_LINE(2)
end SHOW_VALVE_STATUS;

--procedure to check if there is still flow in a system after a valve status change.
--If not, an overpressurization will result.

 procedure CHECK_FOR_FLOW(VALVE_IN: in out VALVE_PTR; FLOW: in out BOOLEAN) is

  COUNT, COUNT2: NATURAL:= 1;

 begin
  VALVE_IN.COUNTED:= TRUE;
  if VALVE_IN.STATUS = OPEN and VALVE_IN.DOWNSTREAM(1) = null then
    FLOW:= TRUE;
  else
    if VALVE_IN.STATUS = OPEN then
      while VALVE_IN.DOWNSTREAM(COUNT) /= null and FLOW = FALSE loop
```

```
        if VALVE_IN.DOWNSTREAM(COUNT).COUNTED = FALSE then
          CHECK_FOR_FLOW(VALVE_IN.DOWNSTREAM(COUNT), FLOW);
        end if;
        COUNT:= COUNT + 1;
      end loop;
    end if;
    if FLOW = FALSE then
      while VALVE_IN.UPSTREAM(COUNT2) /= null and FLOW = FALSE loop
        CHECK_FOR_FLOW(VALVE_IN.UPSTREAM(COUNT2), FLOW);
        COUNT2:= COUNT2 + 1;
      end loop;
    end if;
  end if;

 end CHECK_FOR_FLOW;

--procedure to create the overpressurization if there is no flow

 procedure OVERPRESSURE(AFFECTED: in out VALVE_PTR) is

  COUNT, COUNT2: NATURAL:= 1;

 begin

  AFFECTED.INPUT:= HIGH;
  if TRACE = TRUE then
    SHOW_VALVE_STATUS(AFFECTED);
  end if;
  while AFFECTED.UPSTREAM(COUNT) /= null loop
    AFFECTED.UPSTREAM(COUNT).OUTPUT:= HIGH;
    while AFFECTED.UPSTREAM (COUNT).DOWNSTREAM(COUNT2) /= null loop
      if AFFECTED.UPSTREAM(COUNT).DOWNSTREAM(COUNT2).INPUT /=
                                HIGH then
        OVERPRESSURE(AFFECTED.UPSTREAM(COUNT).DOWNSTREAM
                                (COUNT2));
      end if;
      COUNT2:= COUNT2 + 1;
    end loop;
    if AFFECTED.UPSTREAM(COUNT).STATUS = OPEN then
      OVERPRESSURE(AFFECTED.UPSTREAM(COUNT));
    end if;
    COUNT:= COUNT + 1;
  end loop;
 end OVERPRESSURE;


--procedure to ensure proper propagation of valve input/outputs

 procedure PROPAGATE_VALVE_VALUES (AFFECTED: in out VALVE_PTR) is

  COUNT: NATURAL:= 1;
  FLOW: BOOLEAN:= FALSE;
  CURRENT: VALVE_PTR:= ALPHA_MAIN_STEAM_STOP;

 begin

  if AFFECTED.STATUS = OPEN then
    AFFECTED.OUTPUT:= AFFECTED.INPUT;
```

```
    if TRACE = TRUE then
      SHOW_VALVE_STATUS(AFFECTED);
    end if;
  else
    AFFECTED.OUTPUT:= NONE;
    if TRACE = TRUE then
      SHOW_VALVE_STATUS(AFFECTED);
    end if;
    CHECK_FOR_FLOW(AFFECTED, FLOW);
    if FLOW = FALSE then
      OVERPRESSURE(AFFECTED);
    end if;
    while CURRENT /= null loop
      CURRENT.COUNTED:= FALSE;
      CURRENT:= CURRENT.NEXT;
    end loop;
  end if;
  while AFFECTED.DOWNSTREAM(COUNT) /= null loop
    AFFECTED.DOWNSTREAM(COUNT).INPUT:= AFFECTED.OUTPUT;
    PROPAGATE_VALVE_VALUES(AFFECTED.DOWNSTREAM(COUNT));
    COUNT:= COUNT + 1;
  end loop;

end PROPAGATE_VALVE_VALUES;

--start updating boiler status once a valve propagation has been completed

procedure ALPHA_BOILER_UPDATE is

begin

  ALPHA_BOILER.FUEL_PRESSURE:= ALPHA_FUEL_MANIFOLD.OUTPUT;
  ALPHA_BOILER.AIR:= ALPHA_AIR_REGISTERS.OUTPUT;
  if ALPHA_BOILER.FUEL_PRESSURE = NONE then
    ALPHA_BOILER.FIRE_STATUS:= FALSE;
  end if;
  if ALPHA_BOILER.FIRE_STATUS = FALSE AND GONE_THRU = FALSE then
    ALPHA_BOILER.STEAM_DRUM_PRESSURE:= LOW;
    ALPHA_BOILER.SUPERHEATER_OUTLET_TEMP:= LOW;
    if ALPHA_MAIN_STEAM_STOP.STATUS = OPEN
              or ALPHA_DESUPERHEATER_INLET.STATUS = OPEN then
      REPROPAGATE:= TRUE;
    end if;
    GONE_THRU:= TRUE;
  end if;
  case ALPHA_MANUAL_CHECK_VLV.OUTPUT is
    when NORM=>
      ALPHA_BOILER.WATER_LEVEL:= NORM;
    when NONE=>
      ALPHA_BOILER.WATER_LEVEL:= LOW_ALARM;
    when LOW=>
      ALPHA_BOILER.WATER_LEVEL:= LOW_BUT_NOT_ALARM;
    when HIGH=>
      if ALPHA_MAIN_STEAM_STOP.STATUS = SHUT and
            ALPHA_DESUPERHEATER_INLET.STATUS = SHUT then
        ALPHA_BOILER.WATER_LEVEL:= HIGH_ALARM;
      else
```

```
            ALPHA_BOILER.WATER_LEVEL:= HIGH_BUT_NOT_ALARM;
          end if;
        when others=>
          null;
      end case;
      if VIRTUAL_SH_OUTLET.INPUT = HIGH then
        ALPHA_BOILER.STEAM_DRUM_PRESSURE:= SAFETIES_LIFT_HIGH;
      end if;
      ALPHA_SAFETIES.INPUT:= ALPHA_BOILER.STEAM_DRUM_PRESSURE;
      if ALPHA_SAFETIES.INPUT = SAFETIES_LIFT_HIGH then
        ALPHA_SAFETIES.STATUS:= OPEN;
      else
        ALPHA_SAFETIES.STATUS:= SHUT;
      end if;
      if ALPHA_BOILER.STEAM_DRUM_PRESSURE = SAFETIES_LIFT_HIGH and
              (ALPHA_MAIN_STEAM_STOP.STATUS = OPEN or
              ALPHA_DESUPERHEATER_INLET.STATUS = OPEN) then
        if ALPHA_BOILER.FIRE_STATUS = FALSE then
          ALPHA_BOILER.STEAM_DRUM_PRESSURE:= LOW;
        else
          ALPHA_BOILER.STEAM_DRUM_PRESSURE:= NORM;
        end if;
        REPROPAGATE:= TRUE;
      end if;
      VIRTUAL_SH_OUTLET.INPUT:= ALPHA_BOILER.STEAM_DRUM_PRESSURE;

    end ALPHA_BOILER_UPDATE;

  --procedure to display current or revised status of fireroom equipment

    procedure STATUS_DISPLAY is

    begin

      if REVISED then
        PUT_LINE("The following is the revised status of your plant:");
      else
        PUT_LINE("The following is the current status of your plant:");
      end if;
      NEW_LINE(3);
      SET_COL(40);
      PUT("1A BOILER");
      SET_COL(40);
      PUT("---------");
      NEW_LINE;
      PUT("WATER LEVEL");
      SET_COL(40);
      PUT(ALPHA_BOILER.WATER_LEVEL);
      NEW_LINE;
      PUT("STEAM DRUM PRESSURE");
      SET_COL(40);
      PUT(ALPHA_BOILER.STEAM_DRUM_PRESSURE);
      NEW_LINE;
      PUT("SUPERHEATER OUTLET TEMP");
      SET_COL(40);
      PUT(ALPHA_BOILER.SUPERHEATER_OUTLET_TEMP);
      NEW_LINE;
      PUT("FUEL PRESSURE");
```

```
    SET_COL(40);
    PUT(ALPHA_BOILER.FUEL_PRESSURE);
    NEW_LINE;
    PUT("COMBUSTION AIR");
    SET_COL(40);
    PUT(ALPHA_BOILER.AIR);
    NEW_LINE;
    PUT("STATUS OF FIRES");
    SET_COL(40);
    if ALPHA_BOILER.FIRE_STATUS then
      PUT("LIT");
    else
      PUT("NOT LIT");
    end if;
    NEW_LINE(2);

  end STATUS_DISPLAY;

--main propagation procedure

  procedure PROPAGATE (VALVE_IN: in out VALVE_PTR) is

    CURRENT: VALVE_PTR:= ALPHA_MAIN_STEAM_STOP;

  begin

    STATUS_DISPLAY;
    PUT("Following is a valve status list.");
    NEW_LINE(2);
    PUT("VALVE");
    SET_COL(20);
    PUT_LINE("STATUS");
    while CURRENT /= null loop
      PUT(CURRENT.VALVE_ID);
      SET_COL(20);
      PUT(CURRENT.STATUS);
      NEW_LINE;
      CURRENT:= CURRENT.NEXT;
    end loop;
    NEW_LINE(2);
    PUT(VALVE_IN.VALVE_ID);
    PUT_LINE(" status has been changed.  Propagation underway.");
    NEW_LINE;
    if VALVE_IN.STATUS = OPEN then
      VALVE_IN.STATUS:= SHUT;
      VALVE_IN.OUTPUT:= NONE;
    else
      VALVE_IN.STATUS:= OPEN;
    end if;
    PROPAGATE_VALVE_VALUES(VALVE_IN);
    ALPHA_BOILER_UPDATE;
    REVISED:= TRUE;
    STATUS_DISPLAY;
    if REPROPAGATE = TRUE then
      PROPAGATE_VALVE_VALUES(VIRTUAL_SH_OUTLET);
    end if;
    REPROPAGATE:= FALSE;

  end PROPAGATE;
```

```
--start the main procedure here

begin

  INITIALIZE_PLANT;
  PROPAGATE_VALVE_VALUES(VIRTUAL_SH_OUTLET);
  PROPAGATE_VALVE_VALUES(ALPHA_AIR_SHUTTERS);
  PROPAGATE_VALVE_VALUES(FO_SVC_PUMP_DISCH);
  PROPAGATE_VALVE_VALUES(ALPHA_FEED_STOP_VALVE);
  TRACE:= TRUE;
  PROPAGATE(ALPHA_FUEL_OIL_CONTROL_VALVE);
  PROPAGATE(ALPHA_DESUPERHEATER_INLET);
  PROPAGATE(ALPHA_MAIN_STEAM_STOP);
end BOILER_MODEL;
```

LISP CODE

```
(defstruct valve
   valve-id
   upstream
   downstream
   counted
   next
   status
   input
   output)

(defvar *virtual-sh-outlet* (make-valve
                        :upstream nil
                        :downstream '(*alpha-main-steam-stop*
                                      *alpha-desuperheater-inlet*)
                        :status 'open
                        :counted 'false))

(defvar *alpha-main-steam-stop* (make-valve
                              :valve-id 'a-mss
                              :upstream '(*virtual-sh-outlet*)
                              :downstream '(*er-bulkhead-stop*
                                            *amr-bulkhead-stop*
                                            *mfp-stm-supply*)
                              :counted 'false))

(defvar *alpha-aux-steam-stop* (make-valve
                             :valve-id 'a-ass
                             :upstream '(*alpha-desuperheater-inlet*)
                             :downstream '(*augmentor* *fdb-inlet*
                                           *one-fifty-inlet*
                                           *soot-blower-supply* *pmac-inlet*)
                             :counted 'false))

(defvar *alpha-desuperheater-inlet* (make-valve
                                :valve-id 'a-desup-in
                                :upstream '(*virtual-sh-outlet*)
                                :downstream '(*alpha-aux-steam-stop*
                                              *alpha-bleeder*)
                                :counted 'false))

(defvar *alpha-superheater-protection-inlet* (make-valve
                                          :valve-id 'a-sh-prot-in
                                          :upstream nil
                                          :downstream nil
                                          :counted 'false))

(defvar *alpha-safeties* (make-valve
                      :valve-id 'a-safeties
                      :upstream nil
                      :downstream nil
                      :counted 'false))
```

```
(defvar *alpha-air-registers* (make-valve
                        :valve-id 'a-air-reg
                        :upstream '(*alpha-air-shutters*)
                        :downstream nil
                        :counted 'false))

(defvar *alpha-air-shutters* (make-valve
                       :valve-id 'a-shutters
                       :upstream nil
                       :downstream '(*alpha-air-registers*)
                       :counted 'false))

(defvar *alpha-fuel-manifold* (make-valve
                         :valve-id 'a-fuel-man
                         :upstream '(*alpha-fuel-oil-control-valve*)
                         :downstream nil
                         :counted 'false))

(defvar *alpha-fuel-oil-control-valve* (make-valve
                                  :valve-id 'a-focv
                                  :upstream '(*fo-svc-pump-disch*)
                                  :downstream '(*alpha-fuel-manifold*
                                                *alpha-fuel-recirc*)
                                  :counted 'false))

(defvar *alpha-fuel-recirc* (make-valve
                       :valve-id 'a-fo-recirc
                       :upstream '(*alpha-fuel-oil-control-valve*
                                   *alpha-fuel-manifold*)
                       :downstream nil
                       :counted 'false))

(defvar *fo-svc-pump-disch* (make-valve
                         :valve-id 'fuel-pump-disch
                         :upstream nil
                         :downstream '(*alpha-fuel-oil-control-valve*)
                         :counted 'false))

(defvar *alpha-feed-stop-valve* (make-valve
                           :valve-id 'a-feed-stop
                           :upstream nil
                           :downstream '(*alpha-fwcv*)
                           :counted 'false))

(defvar *alpha-fwcv* (make-valve
                 :valve-id 'a-fwcv
                 :upstream '(*alpha-feed-stop-valve*)
                 :downstream '(*alpha-manual-check-vlv*)
                 :counted 'false))

(defvar *alpha-manual-check-vlv* (make-valve
                            :valve-id 'a-man-chk
                            :upstream '(*alpha-fwcv*)
                            :downstream nil
                            :counted 'false))
```

```
(defvar *alpha-bleeder* (make-valve
                         :valve-id 'a-bleed
                         :upstream '(*alpha-desuperheater-inlet*)
                         :downstream nil
                         :counted 'false))

(defvar *augmentor* (make-valve
                     :valve-id 'aug
                     :upstream '(*alpha-aux-steam-stop*)
                     :downstream nil
                     :counted 'false))

(defvar *fdb-inlet* (make-valve
                   :valve-id 'fdb-in
                   :upstream '(*alpha-aux-steam-stop*)
                   :downstream nil
                   :counted 'false))

(defvar *one-fifty-inlet* (make-valve
                           :valve-id 'one-fifty-in
                           :upstream '(*alpha-aux-steam-stop*)
                           :downstream nil
                           :counted 'false))

(defvar *soot-blower-supply* (make-valve
                              :valve-id 'soot-blow-sup
                              :upstream '(*alpha-aux-steam-stop*)
                              :downstream nil
                              :counted 'false))

(defvar *pmac-inlet* (make-valve
                      :valve-id 'pmac-in
                      :upstream '(*alpha-aux-steam-stop*)
                      :downstream nil
                      :counted 'false))

(defvar *er-bulkhead-stop* (make-valve
                            :valve-id 'er-blkhd-stop
                            :upstream '(*alpha-main-steam-stop*)
                            :downstream nil
                            :counted 'false))

(defvar *amr-bulkhead-stop* (make-valve
                             :valve-id 'amr-blkhd-stop
                             :upstream '(*alpha-main-steam-stop*)
                             :downstream nil
                             :counted 'false))

(defvar *mfp-stm-supply* (make-valve
                          :valve-id 'mfp-stm-sup
                          :upstream '(*alpha-main-steam-stop*)
                          :downstream nil
                          :counted 'false))
```

```
(defstruct boiler
        water-level
        steam-drum-pressure
        superheater-outlet-temp
        fuel-pressure
        air
        fire-status)

(defvar *alpha-boiler* (make-boiler
                        :water-level 'norm
                        :steam-drum-pressure 'low
                        :superheater-outlet-temp 'low
                        :fuel-pressure 'none
                        :air 'none
                        :fire-status 'false))

(defun light-off ()
    (setf (boiler-steam-drum-pressure *alpha-boiler*) 'norm)
    (setf (boiler-superheater-outlet-temp *alpha-boiler*) 'norm)
    (setf (boiler-fuel-pressure *alpha-boiler*) 'norm)
    (setf (boiler-air *alpha-boiler*) 'norm)
    (setf (boiler-fire-status *alpha-boiler*) 'true))

(defun initialize-plant ()
    (light-off)
    (setf (valve-input *virtual-sh-outlet*) 'norm)
    (setf (valve-status *alpha-main-steam-stop*) 'open)
    (setf (valve-next *alpha-main-steam-stop*) *alpha-aux-steam-stop*)
    (setf (valve-status *alpha-aux-steam-stop*) 'open)
    (setf (valve-next *alpha-aux-steam-stop*) *alpha-desuperheater-inlet*)
    (setf (valve-status *alpha-desuperheater-inlet*) 'open)
    (setf (valve-next *alpha-desuperheater-inlet*)
        *alpha-superheater-protection-inlet*)
    (setf (valve-status *alpha-superheater-protection-inlet*) 'shut)
    (setf (valve-next *alpha-superheater-protection-inlet*)
        *alpha-safeties*)
    (setf (valve-input *alpha-superheater-protection-inlet*) 'norm)
    (setf (valve-status *alpha-safeties*) 'shut)
    (setf (valve-next *alpha-safeties*) *alpha-air-registers*)
    (setf (valve-input *alpha-safeties*) 'norm)
    (setf (valve-status *alpha-air-registers*) 'open)
    (setf (valve-next *alpha-air-registers*) *alpha-air-shutters*)
    (setf (valve-status *alpha-air-shutters*) 'open)
    (setf (valve-next *alpha-air-shutters*) *alpha-fuel-manifold*)
    (setf (valve-input *alpha-air-shutters*) 'norm)
    (setf (valve-status *alpha-fuel-manifold*) 'open)
    (setf (valve-next *alpha-fuel-manifold*) *alpha-fuel-oil-control-valve*)
    (setf (valve-status *alpha-fuel-oil-control-valve*) 'open)
    (setf (valve-next *alpha-fuel-oil-control-valve*) *alpha-fuel-recirc*)
    (setf (valve-status *alpha-fuel-recirc*) 'shut)
    (setf (valve-next *alpha-fuel-recirc*) *fo-svc-pump-disch*)
    (setf (valve-status *fo-svc-pump-disch*) 'open)
    (setf (valve-next *fo-svc-pump-disch*) *alpha-feed-stop-valve*)
    (setf (valve-input *fo-svc-pump-disch*) 'norm)
    (setf (valve-status *alpha-feed-stop-valve*) 'open)
    (setf (valve-next *alpha-feed-stop-valve*) *alpha-fwcv*)
    (setf (valve-input *alpha-feed-stop-valve*) 'norm)
    (setf (valve-status *alpha-fwcv*) 'open)
```

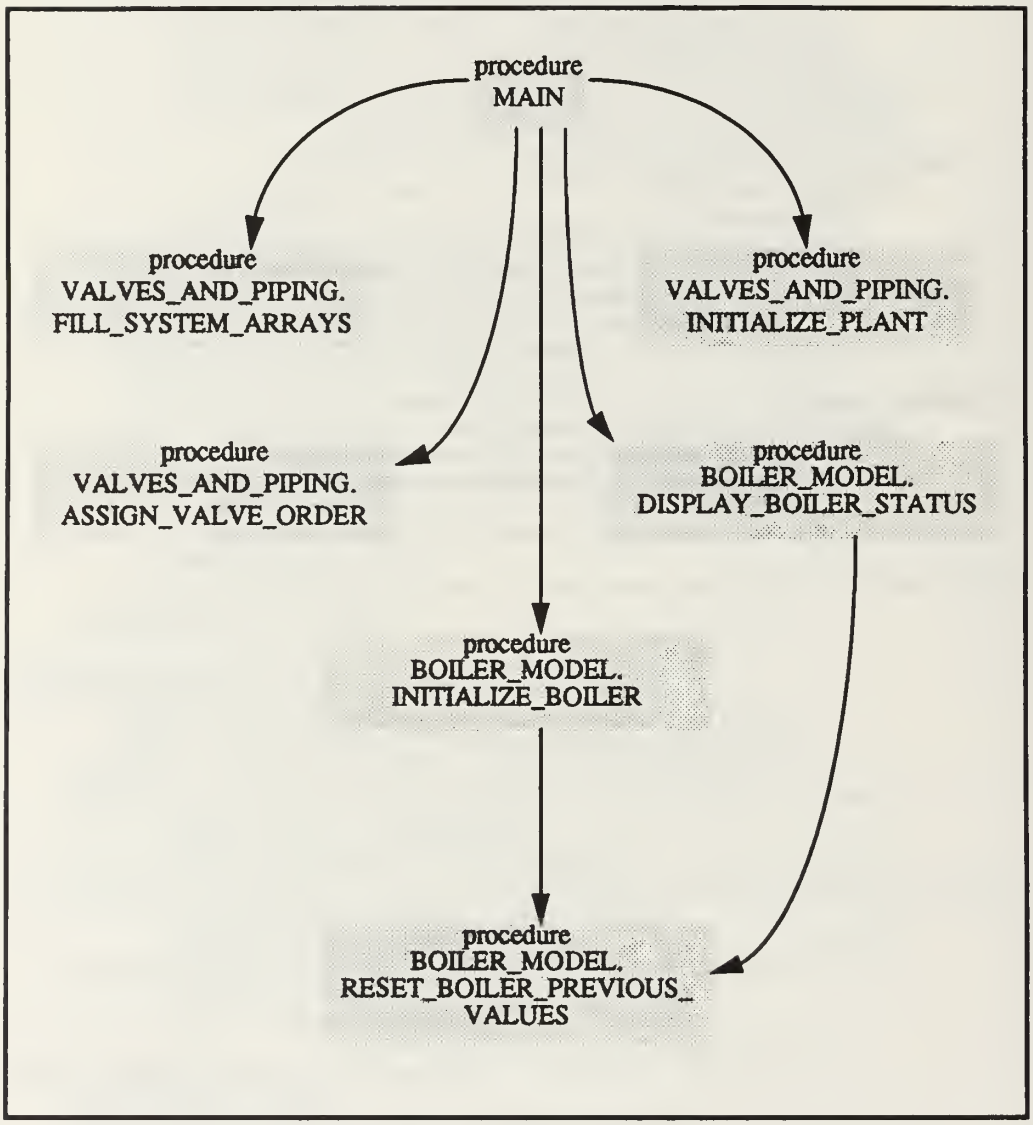
```
        (setf (valve-next *alpha-fwcv*) *alpha-manual-check-vlv*)
        (setf (valve-status *alpha-manual-check-vlv*) 'open)
        (setf (valve-next *alpha-manual-check-vlv*) *alpha-bleeder*)
        (setf (valve-status *alpha-bleeder*) 'shut)
        (setf (valve-next *alpha-bleeder*) *augmentor*)
        (setf (valve-status *augmentor*) 'open)
        (setf (valve-next *augmentor*) *fdb-inlet*)
        (setf (valve-status *fdb-inlet*) 'open)
        (setf (valve-next *fdb-inlet*) *one-fifty-inlet*)
        (setf (valve-status *one-fifty-inlet*) 'open)
        (setf (valve-next *one-fifty-inlet*) *soot-blower-supply*)
        (setf (valve-status *soot-blower-supply*) 'open)
        (setf (valve-next *soot-blower-supply*) *pmac-inlet*)
        (setf (valve-status *pmac-inlet*) 'shut)
        (setf (valve-next *pmac-inlet*) *er-bulkhead-stop*)
        (setf (valve-status *er-bulkhead-stop*) 'open)
        (setf (valve-next *er-bulkhead-stop*) *amr-bulkhead-stop*)
        (setf (valve-status *amr-bulkhead-stop*) 'open)
        (setf (valve-next *amr-bulkhead-stop*) *mfp-stm-supply*)
        (setf (valve-status *mfp-stm-supply*) 'open)
        "plant initialized")

(defun show-valve-status (affected)
        (format t
            '~%VALVE~30tSTATUS~40tINPUT~60tOUTPUT~%------~30t------~40t------~60t------
          ~%~%")
        (do ((current *alpha-main-steam-stop* (valve-next current)))
            ((eq current nil) "That's all folks!")
            (format t
              "~a~30t~a~40t~a~60t~a"
              (valve-valve-id current)
              (valve-status current)
              (valve-input current)
              (valve-output current))
            (if (equal (valve-valve-id current) (valve-valve-id (eval affected)))
              (format t
                "~30t**CHANGE**~%")
              (format t
                "~%")))
        (format t
          "------------------------------------------------------------------------------~%")
        (format t
          "//////////////////////////////////////////////////////////////////////////////~%")
(format t
          "------------------------------------------------------------------------------~%"))

(defun check-for-flow (valve-in flow-status)
        (setf (valve-counted (eval valve-in)) 'true)
        (when (or (equal (valve-status (eval valve-in)) 'shut)
                  (not (equal (valve-downstream (eval valve-in)) nil)))
              (if (equal (valve-status (eval valve-in)) 'open)
                (do* ((n 0 (1+ n))
                      (valve-index1 (first (valve-downstream (eval valve-in)))
                                    (nth n (valve-downstream (eval valve-in)))))
                     ((or (equal valve-index1 nil)
                          (equal flow-status 'flow)))
                     (if (equal (valve-counted (eval valve-index1)) 'false)
                       (setf flow-status (check-for-flow valve-index1 flow-status)))))
```

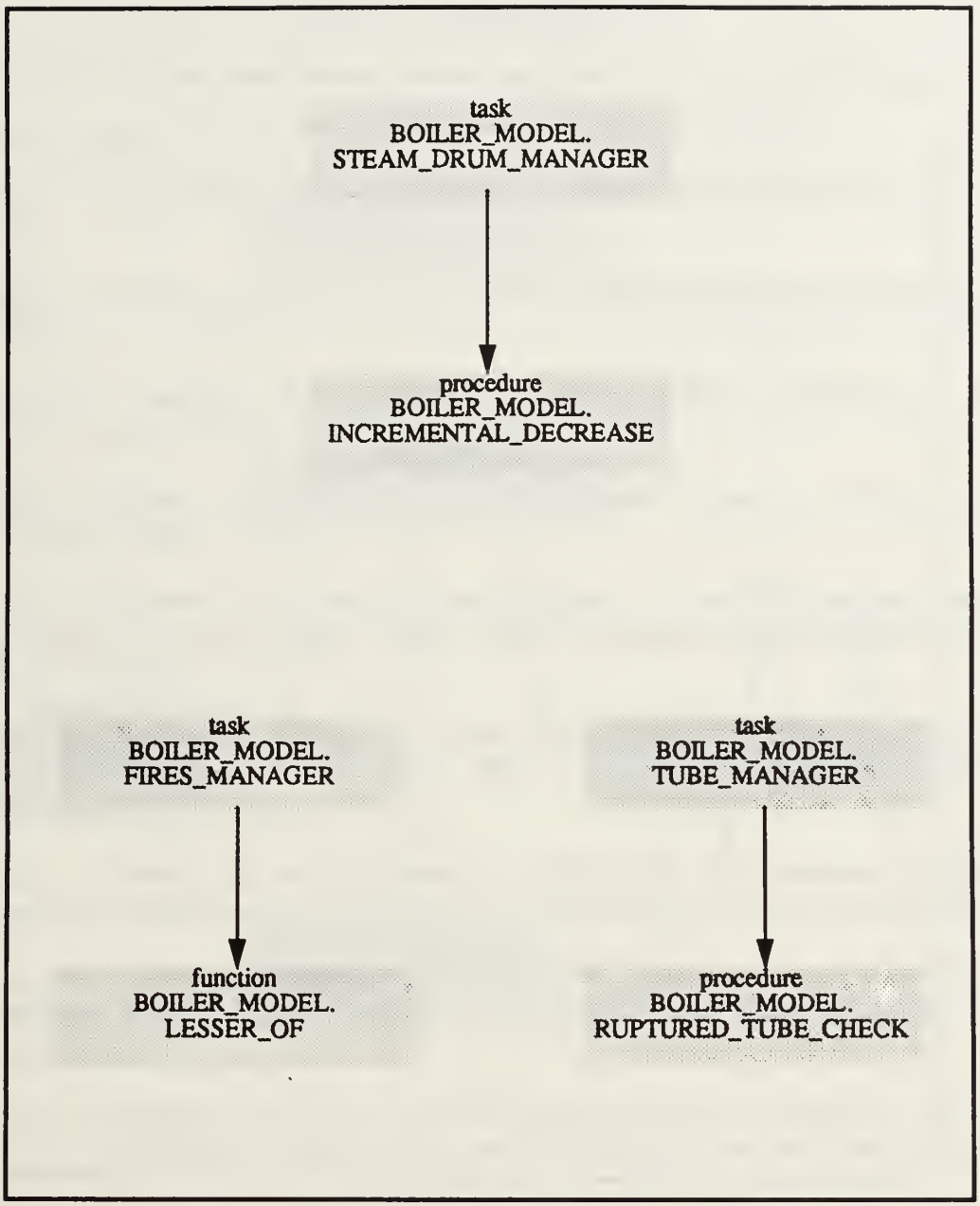
```
            (if (equal flow-status nil)
               (do* ((m 0 (1+ m))
                     (valve-index2 (first (valve-upstream (eval valve-in)))
                                   (nth m (valve-upstream (eval valve-in)))))
                    ((or (equal valve-index2 nil)
                         (equal flow-status 'flow)))
                    (setf flow-status (check-for-flow valve-index2 flow-status)))))
        (if (and (equal (valve-status (eval valve-in)) 'open)
                 (equal (valve-downstream (eval valve-in)) nil))
            (setf flow-status 'flow))
        flow-status)

(defun overpressure (affected trace)
        (setf (valve-input (eval affected)) 'high)
        (if (equal trace 'true)
            (show-valve-status affected))
        (do* ((n 0 (1+ n))
              (valve-index (first (valve-upstream (eval affected)))
                           (nth n (valve-upstream (eval affected)))))
             ((equal valve-index nil))
             (setf (valve-output (eval valve-index)) 'high)
             (do* ((m 0 (1+ m))
                   (valve-index2 (first (valve-downstream (eval valve-index)))
                                 (nth m (valve-downstream (eval valve-index)))))
                  ((equal valve-index2 nil))
                  (if (not (equal (valve-input (eval valve-index2)) 'high))
                      (overpressure valve-index2 trace)))
             (if (equal (valve-status (eval valve-index)) 'open)
                 (overpressure valve-index trace))))

(defun propagate-valve-values (affected trace)
        (let (flow-status)
             (when (equal (valve-status (eval affected)) 'open)
                   (setf (valve-output (eval affected)) (valve-input (eval affected)))
                   (if (equal trace 'true)
                       (show-valve-status affected)))
             (when (equal (valve-status (eval affected)) 'shut)
                   (setf (valve-output (eval affected)) 'none)
                   (if (equal trace 'true)
                       (show-valve-status affected))
                   (if (equal (check-for-flow affected flow-status) nil)
                       (overpressure affected trace))
                   (do ((current *alpha-main-steam-stop* (valve-next (eval current))))
                       ((equal current nil))
                       (setf (valve-counted (eval current)) 'false)))
             (do* ((n 0 (1+ n))
                   (valve-index (first (valve-downstream (eval affected)))
                                (nth n (valve-downstream (eval affected)))))
                  ((equal valve-index nil))
                  (setf (valve-input (eval valve-index)) (valve-output (eval affected)))
                  (propagate-valve-values valve-index trace))))

(defun alpha-boiler-update (gone-thru)
        (let (repropagate)
             (setf (boiler-fuel-pressure *alpha-boiler*)
                   (valve-output *alpha-fuel-manifold*)
                   (boiler-air *alpha-boiler*)
                   (valve-output *alpha-air-registers*))
```

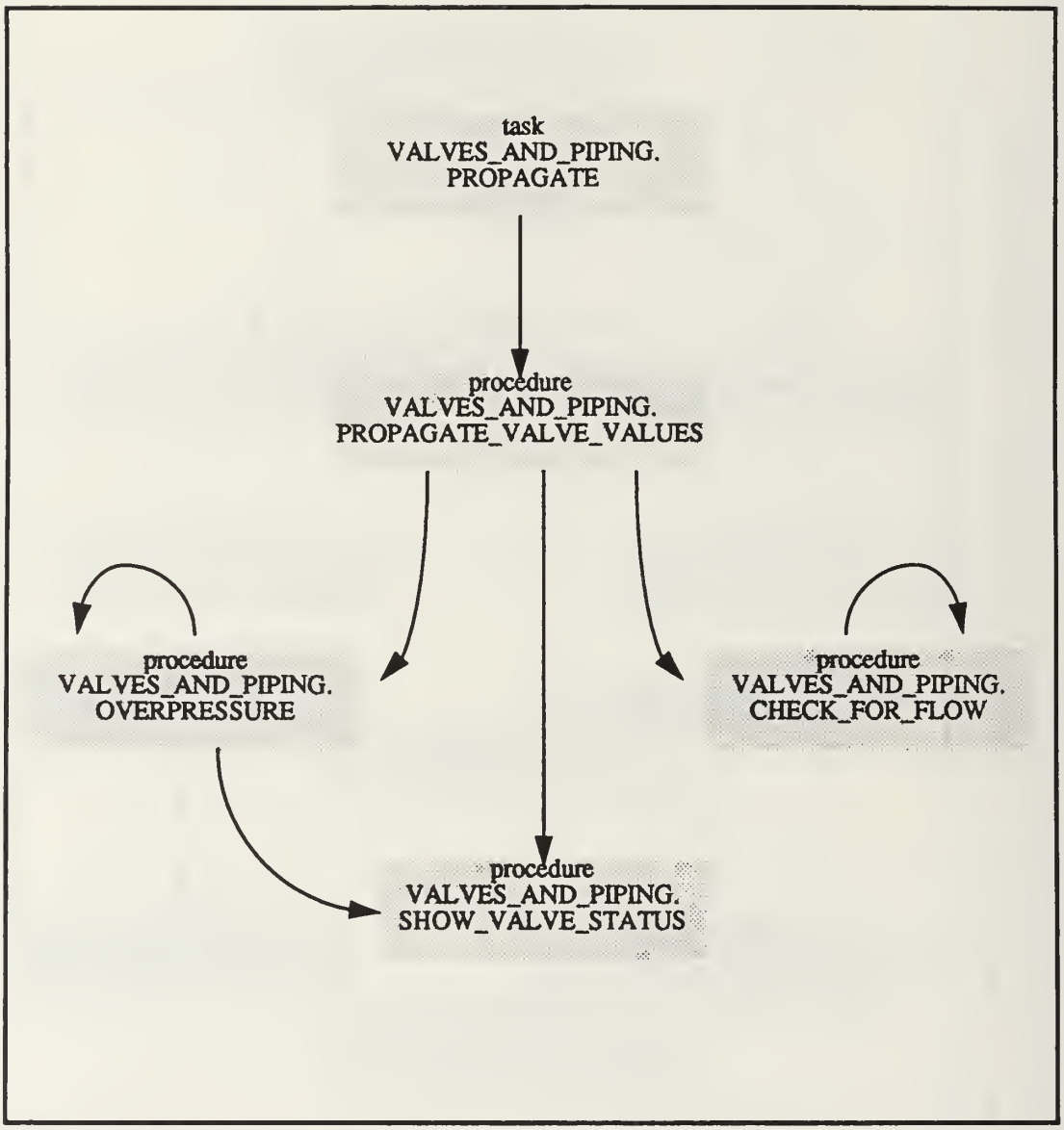
```
        (if (equal (boiler-fuel-pressure *alpha-boiler*) 'none)
            (setf (boiler-fire-status *alpha-boiler*) 'false))
        (when (and (equal (boiler-fire-status *alpha-boiler*) 'false)
                   (equal gone-thru 'false))
              (setf (boiler-steam-drum-pressure *alpha-boiler*) 'low
                    (boiler-superheater-outlet-temp *alpha-boiler*) 'low)
              (if (or (equal (valve-status *alpha-desuperheater-inlet*) 'open)
                      (equal (valve-status *alpha-main-steam-stop*) 'open))
                  (setf repropagate 'true))
              (setf gone-thru 'true))
        (case (valve-output *alpha-manual-check-vlv*)
              ('norm (setf (boiler-water-level *alpha-boiler*) 'norm))
              ('none (setf (boiler-water-level *alpha-boiler*) 'low-alarm))
              ('low (setf (boiler-water-level *alpha-boiler*) 'low-but-not-alarm))
              ('high (if (and (equal (valve-status *alpha-desuperheater-inlet*) 'shut)
                              (equal (valve-status *alpha-main-steam-stop*) 'shut))
                         (setf (boiler-water-level *alpha-boiler*) 'high-alarm)
                         (setf (boiler-water-level *alpha-boiler*) 'high-but-no-alarm)))
              (otherwise nil))
        (if (equal (valve-input *virtual-sh-outlet*) 'high)
            (setf (boiler-steam-drum-pressure *alpha-boiler*) 'safeties-lift-high))
        (setf (valve-input *alpha-safeties*) (boiler-steam-drum-pressure *alpha-boiler*))
        (if (equal (valve-input *alpha-safeties*) 'safeties-lift-high)
            (setf (valve-status *alpha-safeties*) 'open)
            (setf (valve-status *alpha-safeties*) 'shut))
        (when (and (equal (boiler-steam-drum-pressure *alpha-boiler*) 'safeties-lift-high)
                   (or (equal (valve-status *alpha-desuperheater-inlet*) 'open)
                       (equal (valve-status *alpha-main-steam-stop*) 'open)))
              (if (equal (boiler-fire-status *alpha-boiler*) 'false)
                  (setf (boiler-steam-drum-pressure *alpha-boiler*) 'low
                        (boiler-steam-drum-pressure *alpha-boiler*) 'norm))
              (setf repropagate 'true))
        (setf (valve-input *virtual-sh-outlet*) (boiler-steam-drum-pressure *alpha-boiler*))
    (list repropagate gone-thru)))

(defun status-display(revised)
    (if (equal revised 'true)
        (format t "~%The following is the revised status of your plant:")
        (format t "~%The following is the current status of your plant:"))
    (format t "~%~%~40t1A BOILER~%~40t---------~%~%")
    (format t "WATER LEVEL~40t~a~%STEAM DRUM PRESSURE~40t~a~%"
            (boiler-water-level *alpha-boiler*)
            (boiler-steam-drum-pressure *alpha-boiler*))
    (format t "SUPERHEATER OUTLET TEMP~40t~a~%FUEL PRESSURE~40t~a~%"
            (boiler-superheater-outlet-temp *alpha-boiler*)
            (boiler-fuel-pressure *alpha-boiler*))
    (format t "COMBUSTION AIR~40t~a~%STATUS OF FIRES~40t~a~%~%"
            (boiler-air *alpha-boiler*)
            (boiler-fire-status *alpha-boiler*)))

(defun propagate (valve-in trace gone-thru)
    (status-display 'false)
    (format t "~%The following is a valve status list:~%~%")
    (do ((current *alpha-main-steam-stop*
                  (valve-next (eval current))))
        ((equal current nil))
        (format t "~a~20t~a~%"
                (valve-valve-id (eval current))
```

```
                (valve-status (eval current))))
        (format t "~%~a status has been changed. Propagation underway~%~%"
                (valve-valve-id valve-in))
        (if (equal (valve-status (eval valve-in)) 'open)
          (setf (valve-status (eval valve-in)) 'shut
                (valve-output (eval valve-in)) 'none)
          (setf (valve-status (eval valve-in)) 'open))
        (propagate-valve-values valve-in trace)
        (let (repropagate)
            (setf repropagate (first (alpha-boiler-update gone-thru))
                  gone-thru (second (alpha-boiler-update gone-thru)))
            (status-display 'true)
            (if (equal repropagate 'true)
              (propagate-valve-values *virtual-sh-outlet* trace)))
        gone-thru)

(defun boiler-model (valve-in trace)
        (let ((gone-thru 'false))
            (initialize-plant)
            (dolist (element (list *virtual-sh-outlet*
                                   *alpha-air-shutters* *fo-svc-pump-disch*
                                   *alpha-feed-stop-valve*))
                    (propagate-valve-values element 'false))
            (propagate valve-in trace gone-thru)))

(defun practice ()
        (propagate *alpha-main-steam-stop* 'true
                (propagate *alpha-desuperheater-inlet* 'true
                (boiler-model *alpha-fuel-oil-control-valve* 'true))))
```

APPENDIX D (MODEL STRUCTURE CHARTS)




task
BOILER_MODEL.
STEAM_DRUM_MANAGER
procedure
BOILER_MODEL.
INCREMENTAL_DECREASE
task
BOILER_MODEL.
FIRES_MANAGER
function
BOILER_MODEL.
LESSER_OF
task
BOILER_MODEL.
TUBE_MANAGER
procedure
BOILER_MODEL.
RUPTURED_TUBE_CHECK


task
VALVES_AND_PIPING.
PROPAGATE
procedure
VALVES_AND_PIPING.
PROPAGATE_VALVE_VALUES
procedure
VALVES_AND_PIPING.
OVERPRESSURE
procedure
VALVES_AND_PIPING.
CHECK_FOR_FLOW
procedure
VALVES_AND_PIPING.
SHOW_VALVE_STATUS

LIST OF REFERENCES

Baker, L., "Ada and AI Join Forces," *AI Expert*, pp. 39-43, April 1987.

Bennet, E., "Object Lessons in Ada," *Embedded Systems Programming*, pp. 33-37, March 1991.

Booch, G., *Software Engineering with Ada*, 2d ed., Benjamin/Cummings Publishing Co., 1978.

Clancey, W., "Viewing Knowledge Bases As Qualitative Models," *IEEE Expert*, v. 4, pp. 9-23, Summer 1989.

Fulton, S. and Pepe, C., "An Introduction to Model-Based Reasoning," *AI Expert*, pp. 48-55, January 1990.

Gaglio, S., Giacomini, M., Ponassi, A., and Ruggiero, C., "An OPS5 Implementation of Qualitative Reasoning About Physical Systems," *Applied Artificial Intelligence*, v. 4, pp. 37-65, 1990.

Gallanti, M., Stefanini, A., Tomada, L., "ODS: A Diagnostic System Based on Qualitative Modeling Techniques," *1989 IEEE Fifth Conference on Artificial Intelligence Applications*, pp. 141-149.

Hollan, J., Hutchins, E., and Weitzman, L., "STEAMER: An Interactive Inspectable Simulation-Based Training System," *The AI Magazine*, pp. 15-27, Summer 1984.

Inui, M., Miyasaka, N., Kawamura, K., and Bourne, J., "Development of a Model-Based Intelligent Training System," *Future Generation Computer Systems*, v. 5, pp. 59-69, August 1989.

Iwasaki, Y., "Qualitative Physics." In *The Handbook of Artificial Intelligence, Vol. IV*, pp. 323-413. Edited by A. Barr, P. Cohen, and E. Feigenbaum, Addison-Wesley, 1989.

Kinnaird, C., ed., *Rube Goldberg vs. The Machine Age*, p. 37, Hastings House, 1968.

Lutcha, J. and Zejda, J., "Knowledge Represented by Mathematical Models for Fault Diagnosis in Chemical Processing Units," *Knowledge Based Systems*, v. 3, pp. 32-35, March 1990.

Propulsion Plant Manual for Frigates, NAVSEA S-9200-AA-MMA-00-0, Naval Sea Systems Command, 1978.

Towne, D., Munro, A., Pizzini, Q., Surmon, D., Coller, L., and Wogulis, J., "Model Building Tools for Simulation-Based Training," *Interactive Learning Environments*, v. 1, pp. 33-50, 1990.

White, B. and Frederiksen, J., "Causal Models As Intelligent Learning Environments for Science and Engineering Education," *Applied Artificial Intelligence*, v. 3, pp. 83-106, 1989.

Whitehead, J. and Roach J., "Hoist: A Second-Generation Expert System Based on Qualitative Physics," *AI Magazine*, v. 11, pp. 108-119, Fall 1990.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	2
Prof. Yuh-jeng Lee Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	6
Commander Surface Warfare Officers School Command NETC Newport, RI 02841	1
LT James F. Stascavage Surface Warfare Officers School Command DH 121 NETC Newport, RI 02841	2

Thesis
S67776 Stascavage
c.1 BoilerModel.